PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ADEQUAÇÃO DE MODELOS ARQUITETURAIS PARA APLICAÇÕES TEMPO-REAL EM SISTEMAS MANY-CORE

GUILHERME AFONSO MADALOZZO

Tese apresentada como requisito parcial à obtenção de grau de Doutor em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre
2017

Ficha Catalográfica

M178a Madalozzo, Guilherme Afonso

Adequação de Modelos Arquiteturais para Aplicações Tempo-Real em Sistemas Many-Core / Guilherme Afonso Madalozzo . – 2017.
111 f.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Sistemas many-core. 2. Modelagem de sistemas many-core. 3. Aplicações tempo-real. 4. Escalonamento. 5. Mapeamento. I. Moraes, Fernando Gehm. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE TESE DE DOUTORADO

Tese intitulada "Adequação de Modelos Arquiteturais para Aplicações Tempo Real em Sistemas Many-Core" apresentada por Guilherme Afonso Madalozzo como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, aprovada em 12 de janeiro de 2017 pela Comissão Examinadora:

Prof. Dr. Fernando Gehm Moraes (PPGCC/PUCRS - Orientador)

Prof. Dr. César Augusto Missio Marcon (PPGCC/PUCRS)

Prof. Dr. Rafael Fraga Garibotti (FACIN/PUCRS)

Prof. Dr. Sandro Rigo (UNICAMP)

Homologada em...../...../....., conforme Ata No. ..... pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: pgcc@pucrs.br

AGRADECIMENTOS

O caminho percorrido para atingir o objetivo de finalizar esta Tese de Doutorado foi longo e complicado. Nesta etapa de minha vida muitos desafios e obstáculos surgiram para eu provar minha força de superação. Esta força eu encontrei em algumas pessoas importantes. Por isso, aqui agradecerei as pessoas que me ajudaram a superar as diversas dificuldades encontradas.

Primeiramente agradeço a Deus por permitir que eu vivenciasse a experiência de completar esta Tese. Também, agradeço aos ensinamentos e, principalmente, aos exemplos de superação deixados pelos meus falecidos avós. Waldemar, Jandira, Ires e Dirceu não presenciaram esta fase de minha vida, mas com certeza estiveram iluminando os meus caminhos.

Quero deixar um grande agradecimento ao professor Moraes. Sem ele, absolutamente nada teria acontecido. Foram seis anos de muitas, mas muitas, conversas. Serei para sempre grato por tudo que fizeste por mim, desde os puxões de orelha até os elogios pela finalização da Tese. Obrigado por me indicar os melhores caminhos a seguir, do mestrado ao doutorado.

A minha família por sempre me apoiar e incentivar, principalmente nos momentos em que eu quase chutei o balde para longe. Muito obrigado pai e mãe por sempre estarem presentes mesmo a 360km de distância. Minha irmã e cunhado, muito obrigado pelos domingos em que fui acolhido para espairecer e esquecer a vida corrida. Aos meus tios e primos por todo apoio recebido. Agradeço, também, a minha noiva Fernanda que soube entender que um ano e meio passa rápido. Muito obrigado por ser o pilar que faltava para eu chegar até o caminho final desta Tese. Muitos dias longes, mas que hoje valeram a pena.

Agradeço aos funcionários do PPGCC por sempre que necessário quebravam grandes galhos para solucionar os problemas de burocracia. Aos professores, meu agradecimento pelo compartilhamento de seus conhecimentos. Aos meus colegas que muitas vezes nos refugiamos no bar, quadra de tênis e futebol. Obrigado colegas do GAPH: Kuentzer, Ruaro, Wachter, Castilhos, Matheus, LHeck, GHeck, André, Caimi, Jean, Matheus, Alemão e a todos os demais. E a FAPERGS por ter possibilitado e financiado esta pesquisa.

ADEQUAÇÃO DE MODELOS ARQUITETURAIS PARA APLICAÇÕES TEMPO-REAL EM SISTEMAS MANY-CORE

RESUMO

A evolução no processo de fabricação de circuitos integrados permitiu o projeto de SoCs na década de 1990, e atualmente o projeto de sistemas multiprocessados em um único chip - MPSoCs (*Multiprocessor System-on-Chip*). Estes dispositivos são amplamente utilizados em sistemas embarcados, dado o poder computacional oferecido pelos mesmos. Aplicações com restrições de tempo-real vêm sendo utilizadas constantemente, sendo um desafio para o projeto de SoCs. O projeto de MPSoCs é altamente complexo. Especificar as características do MPSoC, definir os componentes que compõe o sistema e analisar suas funcionalidades são decisões que podem apresentar alterações ao longo do desenvolvimento do produto. Métodos tradicionais de projeto não favorecem as tomadas de decisões e encarecem o produto, pois requerem simulação em nível de hardware, estando disponível apenas no final do fluxo de projeto. Para solucionar os problemas apresentados pelos métodos tradicionais de projeto, adotou-se a técnica de projeto baseado em plataforma (PBD – *Platform Based Design*). O método de projeto PBD adota a modelagem de plataformas virtuais em nível de sistema possibilitando rápidas simulações, depuração de *software* e reuso de componentes de *hardware*. Esta Tese tem por objetivo realizar estudos e desenvolvimentos em 2 eixos de pesquisa: (1) modelagem de plataformas virtuais com diferentes organizações de memória; (2) estudo de métodos analíticos para mecanismos de *software* em sistemas com restrições de tempo-real. Para a modelagem de plataformas virtuais usa-se as ADLs (*Architecture Description Language*) OVP e ArchC. Neste tema de trabalho, diversas plataformas foram modeladas em diferentes níveis de abstração (de RTL a modelos sem temporização) e com diferentes arquiteturas de memória (compartilhada e distribuída). Com base nas avaliações realizadas em cada arquitetura, adequou-se a plataforma HeMPS para executar aplicações com restrições de tempo-real. Os resultados apresentaram que, com a utilização do mecanismo de escalonamento e do mapeamento RTA propostos, os dados resultantes das aplicações com restrições de tempo-real aconteceram dentro do período de tempo definido pela aplicação. Comparando plataformas com heurísticas de mapeamento e escalonamento presentes na literatura, a plataforma desenvolvida na presente Tese atende as restrições de aplicações Hard-RT, garantindo 100% das restrições resultantes dos casos de testes.

Palavras chave: sistemas many-core; modelagem de sistemas many-core; aplicações tempo-real; escalonamento; mapeamento.

ADAPTATION OF ARCHITECTURAL MODELS FOR REAL-TIME APPLICATIONS IN MANY-CORE SYSTEMS

ABSTRACT

The evolution of integrated circuit manufacturing process allowed the SoC (System-on-Chip) design in the 90's, and currently the design of multiprocessors systems on chip – MPSoCs (Multiprocessor System-on-Chip). Embedded systems use these devices, due to the offered computational power. The MPSoC design is a challenging task. Specify the MPSoC characteristics, define the components that compose the system and analyze their features are decisions that may change over the product development. Traditional design methods do not favor the design space exploration, leading to expensive products due to required hardware simulation at the gate level, which is only available at the end of the design flow. To solve the design problems of traditional methods, Platform Based Design (PBD) techniques is a design choice. The basis of PBD is a virtual platform model, enabling fast simulations, software debugging and reusability of hardware components. This Thesis comprises the study and development in two research axes: (1) modeling of virtual platforms; (2) analytical methods for software heuristics targeting embedded real-time applications. Virtual platforms are modeled by using ADLs (Architecture Description Languages). This work presents the modeling of several virtual platforms, using different abstraction levels (from RTL to untimed models) and memory architectures (shared and distributed). Based on the evaluations performed in each architecture, the HeMPS platform was adapted to execute real-time applications. The results showed that using the proposed scheduling mechanism and RTA mapping, the results meet the constraints defined by the applications. Comparing platforms with mapping and schedule heuristics on literature, the proposed platform met 100% of the restrictions resulting from the test cases.

Keywords: many-core systems; many-core systems modeling; real-time applications; scheduling; mapping.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ABS	Anti Block System
ADL	Architecture Description Language
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BE	Best Effort
BHM	Behavior Models
BRAM	Block RAM
DBT	Dynamic Binary Translation
DCOM	Distributed Component Object Model
DDR2	Double Data Rate
DMA	Direct Memory Access
DMNI	Direct Memory access and Network Interface
DSM	Distributed Shared Memory
DSOC	Distributed System Object Component
DTW	Dynamic Time Warping
DVFS	Dynamic Voltage Frequency Scale
EDK	Embedded Development Kit
ELF	Executable and Linking Format
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FT	Fault Tolerant
GALS	Globally Asynchronous Locally Synchronous
GANOC	Globally Asynchronous Network-on-Chip
GPS	Global Position System
HAL	Hardware Abstraction Layer
HEAT	Hierarchical Energy-Aware Task Mapping
HP	High Performance
HPC	High Performance Computing
HWGA	Hybrid Worst-fit Generic Algorithm
ICM	Innovative CPU Manager
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine
ISS	Instruction Set Simulator
JIT	Just-in-Time
KPN	Kahn Process Network
LEC-DN	Low Energy Consumption – Dependence Neighborhood
LL-SC	Load-Linked Store-Conditional
LST	Least Slack Time
LU	Lower and Upper
MCAPI	Multicore Communication API
MCM	Maximum Cycle Mean
MCVP	Many-Core Virtual Platform
MILP	Mixed-Integer Linear Programming
MIU	Mesh Interface Unit
MoC	Model of Computation
MP	Manager Processor

MP2I-BencH	Multiprocessor Message Passing Interface Benchmark HeMPS
MPEG	Moving Picture Expert Group
MPI	Message Passing Interface
MPSoC	Multiprocessor System-on-Chip
NI	Network Interface
NoC	Network-on-Chip
NPM	Native Programming
NPU	Network Processing Unit
OCP	Open Core Protocol
OCP-IP	Open Core Protocol International Partnership
OP	Open Platform
OpenCL	Open Computing Language
OVP	Open Virtual Platform
PBD	Platform Based Design
PE	Processor Element
PGAS	Partitioned Global Address Space
PPM	Peripherals Model
QNoC	Quality-of-Service NoC
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RMA	Remote Memory Access
RMBF	Rate Monotonic Best Fit
RMFF	Rate Monotonic First Fit
ROI	Return on Investment
RT	Real-time
RTA	Response Time Analysis
RTFA	Real-Time Thermal Feasibility Analysis
RTL	Register Transfer Level
RTOS	Real Time Operating System
SHA	Security Hash Algorithm
SM	Shared Memory
SMP	Symmetrical Multiprocessing
SoC	System-on-Chip
SP	Slave Processor
SPLASH-2	Stanford ParalleL Applications for SHared memory
TALK	Thermal-Aware Leakage
TALO	Temperature-Aware Leakage Optimization
TFA	Thermal Feasibility Analysis
TLM	Transaction Level Modeling
UART	Universal Asynchronous Receiver/Transmitter
VC	Virtual Channel
VMI	Virtual Machine Interface
VOP	Video Object Plane
VOPD	Video Object Plane Decoder
VP	Virtual Platform
WCET	Worst-Case Execution Time

SUMÁRIO

1 INTRODUÇÃO

Sistemas embarcados estão cada vez mais presentes no dia-a-dia dos consumidores, podendo ser facilmente encontrados em produtos como eletrodomésticos, carros, televisores, equipamentos de telecomunicação, telefones celulares e *tablets*, e segundo Aguiar et al. [AGU08] sua principal característica é executar funcionalidades específicas. Com a crescente comercialização destes bens de consumo, aumentou-se rapidamente a fabricação de SoCs (*System-on-Chip*). Segundo Chang et al. [CHA99], SoC é um circuito integrado complexo, em um único *chip,* cujo produto final reúne os principais elementos funcionais de um sistema embarcado.

Sistemas embarcados modernos são compostos por SoCs com múltiplos processadores. Sistemas multiprocessados são denominados MPSoCs (*Multiprocessor System-on-Chip*). Um MPSoC consiste em uma infraestrutura que reúne componentes de *hardware* e *software*. Por exemplo, a infraestrutura de comunicação entre os componentes de *hardware* pode ser uma decisão de projeto independente, podendo ser através de barramentos ou NoCs (*Network-on-Chip*). Porém, a comunicação entre os componentes de *software* é diretamente dependente da infraestrutura em nível de *hardware* – através de memória compartilhada ou por troca de mensagens. Outro exemplo, no nível de computação, aplicações com execução de cálculos com ponto flutuante não podem ser executadas de forma otimizada em processadores sem suporte a tal requisito.

Segundo Santos [SAN13], os componentes de *hardware* de um MPSoC são formados por componentes físicos, como barramentos, módulos de DMA (*Direct Memory Access*), memórias e processadores. Estes componentes podem ser parametrizados de acordo com as necessidades do sistema em desenvolvimento. Os componentes de *software* são modelados através de linguagens de programação, tal como C, Java, Objective-C e Swift.

Projetar MPSoCs é uma tarefa altamente complexa. Tanto os projetistas de *hardware* quanto os projetistas de *software* devem ter um conhecimento global das características do sistema, como funcionalidades específicas do produto final, organização de memória, tamanho de armazenamento e protocolos de comunicação. A tomada de decisões, pelos projetistas, torna-se cada vez mais complexa conforme o aumento do número de elementos no sistema. O projeto tradicional de MPSoCs, através de modelagem RTL (*Register Transfer Level*), é uma importante barreira devido ao elevado tempo de desenvolvimento, simulação e validação, elevando assim o custo final do produto.

Novas técnicas e métodos de projetos devem ser considerados para solucionar os problemas apresentados pelo desenvolvimento tradicional de MPSoCs. Modelagem em nível RTL gera informações precisas que auxiliam os projetistas na tomada de decisões, como desempenho, área e consumo energético. Por outro lado, este tipo de modelagem afeta diretamente o *time-to-market* [VAH01], que define o exato momento para a introdução do produto no mercado, visando

maior lucratividade e retorno do investimento (ROI – *Return on Investment*). Dessa forma, surge a necessidade de métodos que reúnam os requisitos de *time-to-market* e que minimizem os problemas e complexidades de modelagem tradicionais.

Projeto baseado em Plataformas (PBD – *Platform Based Design*) reúne as características necessárias para minimizar os problemas encontrados nos métodos tradicionais de desenvolvimento de SoCs [BOU07]. PBD tem por principal característica a modelagem de MPSoCs em nível de sistema oferecendo otimizações no processo de desenvolvimento, como simulação rápida, alto nível de depuração de *software* e flexibilidade na modelagem de SoCs com o reuso e alteração rápida de componentes. Porém, o uso de modelagem em nível de sistema penaliza a precisão dos resultados, como desempenho e área. Mesmo que algumas informações sejam afetadas, PBD auxilia os projetistas nas tomadas de decisões, como otimização de energia, verificação e depuração do *software* por pontos de parada (*break points*). Dessa forma, PBD reduz os riscos de erros, falhas e defeitos no desenvolvimento do *software*, diminuindo o custo do projeto e possibilitando a finalização do produto em menor tempo do que os métodos tradicionais.

A Figura 1 apresenta o ciclo de desenvolvimento de SoCs com a utilização do método tradicional de projetos e com uso de PBD. Inicialmente, em ambos métodos, tem-se a especificação da plataforma a ser projetada, definindo-se os processadores (ARM, MIPS, PowerPC), infraestrutura de comunicação (barramentos, NoC), tamanho dos blocos da memória, organização de memória (arquitetura de memórias distribuídas ou arquitetura de memória compartilhada), entre outras características necessárias para as funcionalidades do produto. Com a arquitetura da plataforma definida, inicia-se o processo de desenvolvimento do *hardware*. No método tradicional de projeto as etapas são sequenciais, ou seja, o desenvolvimento do *software* só será iniciado depois que a plataforma real estiver totalmente, ou parcialmente (mas funcional), desenvolvida. Por outro lado, PBD apresenta um ciclo paralelo de projeto. Quando a plataforma estiver com sua especificação definida, juntamente com o início do processo de desenvolvimento do *hardware*, é modelada uma plataforma virtual (VP – *Virtual Platform*) contendo os mesmos componentes e os mesmos comportamentos especificados para a plataforma real. Plataformas virtuais são meios eficientes no qual as funcionalidades do *software* e o *hardware* desejado podem ser projetados e verificados em conjunto nas fases iniciais do projeto [REK13]. Dessa forma, tem-se uma integração do *hardware* com o *software* desde o início do processo de desenvolvimento de SoCs. Também, reduz-se o tempo de desenvolvimento do *software* em virtude de que os testes são feitos em nível de sistema, sendo mais rápida a validação das funcionalidades. Por fim, o ciclo de desenvolvimento de SoCs com utilização de PBD é finalizado quando a plataforma real estiver pronta e o *software* testado e validado na mesma.

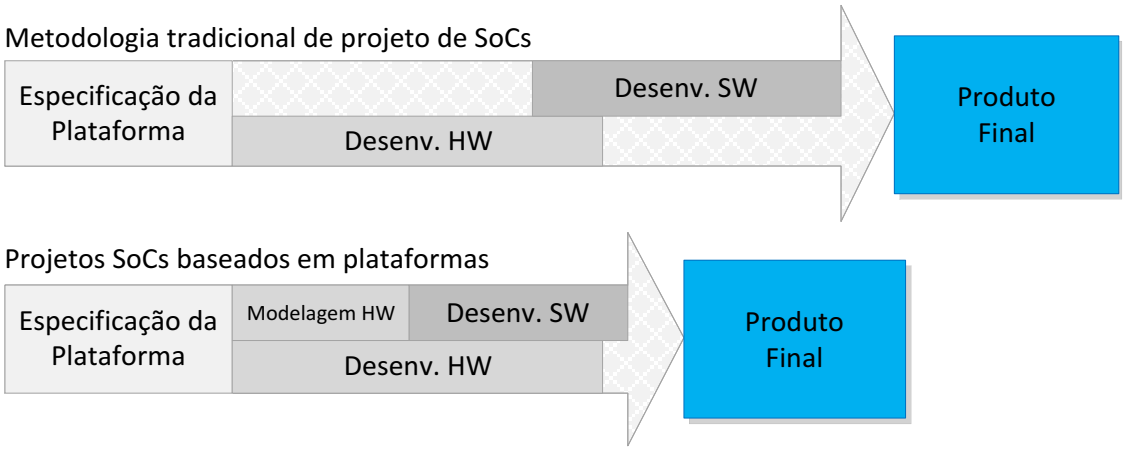


Figura 1 - Método tradicional de projeto de SoCs *vs* PBD. Adaptado de: [SAN04].

Plataformas virtuais são modeladas através de ADLs (*Architecture Description Language*) em nível de sistema, como ArchC [RIG03] e OVP (*Open Virtual Platform*) [OVP15]. ArchC é uma ADL baseada na linguagem SystemC, com recursos que permitem ao projetista explorar, testar e verificar arquiteturas de processadores através do simulador SystemC. OVP é um ambiente de desenvolvimento de plataformas virtuais, proposta pela Imperas [OVP15], que possibilita a modelagem e a simulação de um sistema embarcado, descrito nas linguagens de programação C e C++. A utilização de PBD com ADLs permite explorar MPSoCs com diferentes infraestruturas de comunicação, processadores e arquiteturas de memórias.

Tradicionalmente, o software é dividido em programas de sistema e programas aplicativos. Segundo Ian Sommerville [SOM10], programas de sistemas consistem nos softwares que fazem a interface com o hardware do dispositivo. Já os programas aplicativos são programas escritos para solucionar determinado problema específico. O uso de PBDs possibilita aos engenheiros de hardware e software uma exploração arquitetural capaz de auxiliar na tomada de decisões conforme as características e necessidades do sistema.

Sistemas com características específicas, como sistemas de tempo-real (RT, do inglês, *Real-Time*), necessitam de uma arquitetura que atenda os requisitos de suas aplicações. Segundo Liu [LIU13], sistemas RT são classificados em Hard RT e Soft RT. A definição do tipo de sistema é definida pelas restrições estabelecidas pelas aplicações. Sistemas Hard RT definem que todas as restrições impostas pelas aplicações devem, obrigatoriamente, ser garantidas. Por outro lado, sistemas Soft RT toleram perder ocasionalmente algumas restrições de aplicações RT.

A modelagem de plataformas abstratas possibilita uma avaliação do sistema e a análise das limitações de *software* e *hardware* presentes na arquitetura. Limitações podem surgir em algoritmos de escalonamento e mapeamento que, em determinados momentos, podem comprometer o desempenho das aplicações executando no sistema. Também, a interface de comunicação utilizada pela arquitetura pode influenciar na garantia de requisitos de aplicações, podendo gerar *hotspots* no sistema caso a tomada de decisões não for a correta.

O crescente uso de MPSoCs aumenta a necessidade de avaliar plataformas com diferentes organizações de memórias. Segundo Matilainen et al. [MAT11], aplicações embarcadas estão cada vez mais complexas, e a arquitetura de memória define o modelo de programação adotado pelo MPSoC. O desenvolvimento de *benchmarks* para avaliação de plataformas é altamente dependente das características do MPSoC modelado. Por exemplo, *benchmarks* desenvolvidos com modelo de programação baseado em *threads* se aplicam aos MPSoCs com arquitetura de memória compartilhada, enquanto *benchmarks* baseados em troca de mensagens se adequam em MPSoCs com arquitetura de memórias distribuídas.

Para avaliação e validação de projetos utilizando PBD, o *software* é modelado conforme a arquitetura de memória definida na especificação da plataforma. Modelos de programação disponibilizam um conjunto de funções para descrever as aplicações, através de interfaces de programação (API – *Application Programming Interface*). APIs como POSIX PThread, OpenMP e acPThread [DUE14] são adequadas unicamente para plataformas com arquitetura de memória compartilhada, onde os dados entre as tarefas de aplicações são compartilhados através da memória. Por outro lado, arquiteturas de memórias distribuídas são mais adequadas para APIs como MCAPI (*Multicore Communication API*) [MAT11] e MPI (*Message Passing Interface*), que tem como principal característica a comunicação por troca de mensagens.

1.1 Hipótese a ser demonstrada por esta Tese

Esta Tese busca demonstrar duas hipóteses: (*i*) modelagem de arquiteturas podem ser utilizadas para avaliar limitações de *hardware* para sistemas RT, tais como: tráfego localizado (*hotspot*) e congestionamento da NoC ; e (*ii*) demonstrar que o uso de heurística de mapeamento de tarefas com base em modelos analíticos, juntamente com algoritmos de escalonamento de tarefas podem definir o sistema como *hard* ou *soft* RT.

1.2 Objetivos

Com a intenção de demonstrar as hipóteses definidas acima, o objetivo estratégico desta Tese primeiramente é reunir diferentes técnicas de modelagem de arquiteturas multiprocessadas em diferentes níveis de abstração, tendo por NoC a infraestrutura de comunicação. Com a utilização de destes modelos pode-se analisar MPSoCs com diferentes organizações de memórias, avaliando as interfaces de comunicação para analisar o comportamento do sistema para aplicações com restrições de tempo-real. Tendo uma arquitetura definida dentro das limitações de sistemas RT o segundo objetivo estratégico desta Tese é adaptar um modelo analítico de aplicações de tempo-real como função custo do mapeamento de tarefas, no intuito de garantir que as tarefas não violem os requisitos de tempo de suas aplicações.

Para atingir os objetivos estratégicos, os seguintes objetivos específicos devem ser

cumpridos:

- Modelar a NoC Hermes [MOR04], empregando a linguagem SystemC-TLM2.0, integrando-a à plataforma MPSoCBench (com componentes descritos em ArchC) [DUE14]. Este é um trabalho de cooperação com o Laboratório de Sistemas Computacionais (LSC) do Instituto de Computação da Universidade de Campinas (IC-Unicamp);
- Validar a plataforma MPSoC HeMPS [MAN13] modelada em dois níveis de abstração: RTL e OVP;
- Adaptar benchmarks de aplicações com requisitos de tempo-real para as plataformas desenvolvidas;
- Propor um mapeamento de tarefas com a utilização de um método analítico próprio para sistemas de tempo-real;
- Investigar diferentes técnicas de escalonamento, buscando atender as restrições de tempo impostas pelas aplicações.

1.3 Originalidade e Contribuições da Tese

A originalidade desta Tese é a adequação de modelos arquiteturais para aplicações de tempo-real em sistemas multiprocessados. Esta adequação ocorre pela análise de arquiteturas com diferentes hierarquias de memória, utilizando modelagem abstrata de arquiteturas. Também, a adequação ocorre através do desenvolvimento de novas técnicas de mapeamento e escalonamento. Estas técnicas são validadas em um modelo arquitetural RTL-VHDL sintetizável, com precisão de ciclo de relógio.

A presente Tese apresenta duas principais contribuições: (*i*) modelagem de arquiteturas com diferentes hierarquias de memória em alto nível de abstração, capaz de auxiliar projetistas na tomada de decisão através de simulações rápidas e dados de depuração; (*ii*) propor uma técnica de mapeamento com base em um modelo analítico específico para aplicações com restrições de tempo-real, juntamente com um algoritmo de escalonamento adequado a este tipo de sistema. Esta Seção apresenta as contribuições desta Tese, assim como contribuições de trabalhos realizados pelo Grupo GAPH. A FIGURA posiciona as contribuições desta Tese que a seguir são detalhadas:

- Desenvolvimento de MPSoCs descritos com plataformas virtuais, apresentadas no Capítulo 1. Neste contexto, esta Tese contribui com três novos modelos apresentando diferentes características:
    - HeMPS-ML, apresentada na Seção 3.1, onde foi desenvolvida uma HAL (*Hardware Abstraction Layer*) para que o software execute independentemente do nível de abstração do hardware (RTL-VHDL, RTL-SystemC ou OVP) [MAD15]. Tal plataforma emprega arquitetura de memória distribuída, executando

aplicações com a API MPI embarcada e a gerência da comunicação e de recursos do sistema é feita através do sistema operacional.

  - MPSoCBench, apresentada na Seção 3.2, sistema desenvolvido em alto nível de abstração onde a NoC é descrita em SystemC-TLM2.0 e os periféricos e processadores em ArchC. Essa plataforma emprega arquitetura de memória compartilhada sem a utilização de sistemas operacionais para gerenciar os recursos do sistema. Esta pesquisa foi realizada em cooperação com o Laboratório de Sistemas Computacionais (LSC) do Instituto de Computação da Universidade de Campinas (IC-Unicamp).
  - A terceira plataforma possui as mesmas características da HeMPS-ML, porém desenvolvida em alto nível de abstração (modelagem OVP) e sem sistema operacional. Tal plataforma foi desenvolvida pela necessidade de efetuarmos um comparativo justo da plataforma MPSoCBench com uma plataforma de memórias distribuídas com características de *software* semelhante.

- Desenvolvimento de um *Benchmark Suite* (MP2I-BencH – *Multiprocessor Message Passing Interface Benchmark HeMPS*) para sistemas embarcados com arquitetura de memórias distribuídas, através da API MPI. Para o comparativo, no MPSoCBench (sistema com arquitetura de memória compartilhada) já existem aplicações disponíveis para avaliação. Além do desenvolvimento de novas aplicações para efetuarmos um comparativo entre plataformas, foi desenvolvido um *benchmark* de veículos autônomos [IND14] para utilizarmos como estudo de caso na presente Tese.
- Contribuições referentes à adequação dos modelos arquiteturais para suportar aplicações de tempo-real:
  - Comparar arquiteturas com diferentes hierarquias de memória para analisar as restrições que um sistema de tempo-real deve suportar;
  - Propor um novo algoritmo de mapeamento que utiliza como função custo um método analítico capaz de verificar se uma tarefa de aplicação de tempo-real pode ou não ser executada em determinado processador;
  - Adaptar ao MPSoC de referência o algoritmo de escalonamento preemptivo por prioridades;
  - Validar a técnica de escalonamento comparando-a com outra técnica utilizada em sistemas de tempo-real;
  - Validar a técnica de mapeamento proposta comparando-a com diferentes técnicas e executando em cenários com dezenas de processadores. A validação considera a quantidade de restrições de aplicações RT violadas, considerando a latência das tarefas.

1.4 Estrutura do Documento

O restante deste Documento é organizado como segue. O Capítulo 2 apresenta os trabalhos relacionados à Tese. Tais trabalhos foram divididos em três categorias, que em conjunto reúnem as referências necessárias para a presente Tese: (*i*) plataformas multiprocessadas, com ênfase em modelagens virtuais com uso de ADLs como OVP e ArchC; (*ii*) MPSoCs RT, apresentando métodos analíticos para usar como função custo em técnicas de mapeamento e escalonamento propostos para sistemas de tempo-real; (*iii*) benchmarks utilizados para os testes dos mecanismos propostos nesta Tese. O Capítulo 3 apresenta a modelagem das plataformas multiprocessadas. As plataformas são modeladas em diferentes níveis de abstração e com diferentes organizações de memórias. Estas plataformas permitem avaliar aplicações com modelos de programação distintos (PThread e MPI). As avaliações dos modelos arquiteturais são apresentadas no Capítulo 4. O Capítulo 5 apresenta a modelagem de uma plataforma de tempo-real, apresentando as funções do sistema, seu mecanismo de escalonamento e mapeamento. O Capítulo 6 avalia a plataforma de tempo-real através de um estudo de caso de um conjunto de aplicações com restrições de tempo-real. Por fim, o Capítulo 7 finaliza a Tese concluindo o trabalho e apresentando caminhos para pesquisas futuras.

2 ESTADO DA ARTE

Neste Capítulo apresenta-se o estado da arte nos temas relacionados à Tese. Analisou-se pesquisas realizadas em modelagens de plataformas multiprocessadas, com NoCs como meio de comunicação, e sistemas embarcados de tempo-real.

Os trabalhos citados relacionam-se com os objetivos apresentados no Capítulo anterior: (*i*) modelagem de plataformas multiprocessadas; (*ii*) plataformas com características de tempo-real; e (*iii*) benchmarks. Este Capítulo é organizado como segue. Na Seção 2.1 apresentam-se os trabalhos relacionados a plataformas multiprocessadas, com ênfase em descrições voltadas para plataformas virtuais (OVP e ArchC), que são as ADLs (*Architecture Description Language*) utilizadas na presente Tese. Na Seção 2.2 têm-se os trabalhos relacionados a mecanismos de mapeamento e escalonamento para MPSoCs que executam aplicações com restrição de tempo-real. Na Seção 2.3 apresentam-se os trabalhos relacionados aos benchmarks, apresentando as aplicações adaptadas no presente trabalho. Ao final de cada seção apresenta-se o comparativo entre os trabalhos relacionados e o posicionamento da Tese em relação a estes trabalhos.

2.1 Plataformas Multiprocessadas

Nesta Seção apresenta-se o estado da arte na área de MPSoCs. São analisados trabalhos em diferentes níveis de abstração, com diferentes métodos para modelagem, como RTL sintetizável, SystemC, OVP e ArchC.

2.1.1 Rekik et al.

Rekik et al. [REK13] apresentam uma exploração da capacidade de implementação e simulação de MPSoCs com OVP, focando na facilidade de comunicação entre processadores. Os Autores comentam que a comunicação e compartilhamento de dados entre múltiplos PEs (Elementos de Processamento) é um grande desafio em projetos de MPSoCs. A topologia e organização de memória podem ser compartilhada, distribuída ou distribuída compartilhada. Os Autores desenvolveram diversas arquiteturas multiprocessadas homogêneas usando diferentes tipos de processadores e duas hierarquias de memória, sendo a comunicação realizada através da memória compartilhada.

Para demonstrar a eficiência do emprego de OVP, foram testadas diferentes arquiteturas multiprocessadas. Com uso de plataformas virtuais, o desenvolvimento do *software* e do *hardware* podem progredir paralelamente. Primeiramente, os Autores definem uma arquitetura composta por dois processadores (homogêneos) e três memórias (duas locais e uma compartilhada), onde cada processador é conectado em uma memória local e os dois conectam-se à memória compartilhada. Nesse cenário, as aplicações são armazenadas na memória compartilhada. A segunda implementação apresentada pelos Autores é uma arquitetura com estrutura similar à

primeira implementação, porém as aplicações são armazenadas nas memórias locais. A terceira implementação é uma arquitetura de memória compartilhada com comunicação entre diferentes processadores executando um sistema operacional Linux embarcado.

Como resultados os Autores concluem que o OVP é uma ADL que permite a criação de plataformas tanto homogêneas quanto heterogêneas, apenas instanciando diferentes tipos de processadores e modelos computacionais. Também, descrevem a rápida modelagem e reusabilidade de componentes.

2.1.2 Beserra et al.

Beserra et al. [BES12] propõem uma integração de componentes virtuais, descritos em TLM, seguindo um determinado MoC (*Model of Computation*). Os Autores utilizam a plataforma ForSyDe [SAN04] que disponibiliza bibliotecas e ferramentas para diferentes MoCs. Entre as principais contribuições do trabalho, destacam-se: (*i*) um conjunto de *wrappers* que possibilitam a integração entre modelos TLM e processos modelados no ForSyDe; (*ii*) uma abordagem em que é possível aumentar o nível de abstração do modelo.

É utilizado o nível de abstração TLM-*untimed*, onde todos os processos executam em paralelo para acessar os recursos do sistema no mesmo instante de tempo. TLM é baseado em transações que podem ser definidas por diversos protocolos de comunicação, com diferentes semânticas. Para os testes, são apresentados dois diferentes protocolos: protocolo básico entre processos TLM; e protocolo com adição de *wrappers* para comunicação entre TLM e ArchC.

A avaliação do protocolo de comunicação básica da biblioteca TLM é apresentada por Rose et al. [ROS05], mostrando separadamente a camada de transporte TLM da camada de aplicação. Os pacotes que trafegam pela rede consistem de um *request* e um *response* com endereçamento e dados configurados na camada de aplicação. No protocolo TLM, onde há comunicação com modelos ArchC, todas as instruções são executadas corretamente, mas sem possuir informação temporal. O protocolo implementado é similar ao primeiro caso apresentado. Quando o simulador faz um *request,* o ArchC cria um pacote que contém o endereço e executa a função de transporte *slave* através do canal TLM. Os Autores relatam que o desempenho da plataforma com utilização de *wrappers* e ArchC foi maior do que com a utilização de canais SystemC-TLM padrão.

2.1.3 Zhang et al.

Zhang et al. [ZHA13] apresentam uma ferramenta de modelagem, MCVP-NoC (*Many-Core Virtual Platform with Network-on-Chip*), projetada para MPSoCs de grande escala. O MCVP-NoC foi desenvolvido em SystemC-TLM2.0. Uma camada em OVP foi integrada para prover um rápido tempo de simulação dos processadores, memória e barramento. Também, o simulador Orion2 foi integrado para prover estimativa de dissipação de potência e área. Todos componentes OVP podem ser conectados por uma interface TLM2.0.

Quando a simulação do MCVP-NoC é finalizada, além das informações de desempenho, informações de potência e área são reportados. A integração OVP possibilita alto desempenho de simulação, análise e validação das plataformas. Os Autores desenvolveram uma ferramenta para geração automática da NoC, que é gerada com base em um arquivo de configuração de plataforma.

Os Autores reportam que os resultados simulados em alto nível de abstração (plataformas virtuais) alcançaram um ganho de 40 vezes em relação à modelagem RTL. O MCVP-NoC pode estimar o desempenho do sistema, assim como área e potência, e pode ser usado para auxiliar na exploração do espaço de projeto.

2.1.4 Lukovic et al.

Lukovic et al. [LUK08] apresentam um framework capaz de reconfigurar sistemas multiprocessados baseados em NoC. Para isso, os Autores propõem modificações na ferramenta Xilinx EDK (*Embedded Development Kit*) para dar suporte à geração automática desse tipo de MPSoCs. Esse framework possibilita que projetistas possam desenvolver protótipos rapidamente. Assim, a proposta permite testar, verificar e depurar o sistema projetado, reduzindo o tempo final de desenvolvimento.

Como elemento de processamento, os Autores utilizam o processador MicroBlaze fornecido pela Xilinx. O MicroBlaze é um processador com arquitetura RISC (*Reduced Instruction Set Computer*), otimizado para implementações em FPGAs da Xilinx. Ao MicroBlaze é conectada uma memória local via barramento. O sistema proposto pelos Autores contém memórias compartilhadas, implementadas usando partes da BRAM (*Block RAM*) disponíveis nas FPGAs da Xilinx.

Para obter os resultados, os Autores sintetizaram em um dispositivo Xilinx Virtex-II dois cenários diferentes. O primeiro foi um sistema homogêneo com três MicroBlazes interconectados via NoC. O segundo foi um sistema com arquitetura de memória compartilhada de 16KB com dois MicroBlazes, todos componentes interconectados via NoC. O resultado final do mecanismo de automação (EDK proposto pelos Autores) é um arquivo *bitstream* (sequência de bits) configurável, que é diretamente carregado na FPGA.

2.1.5 Benini et al.

Benini et al. [BEN12] apresentam o projeto da Plataforma 2012 (P2012) ao qual utiliza-se banco de memórias compartilhadas para interconectar os elementos de processamento. P2012 pode executar aplicações paralelas com padrões OpenCL (*Open Computation Language*) e OpenMP [DAG98]. Uma variedade de plataformas virtuais é disponibilizada para dar suporte a facilidades no desenvolvimento de software, maior desempenho na análise e otimização de resultados, e melhor exploração na interação *hardware*/*software*.

Os Autores apresentam o P2012 como uma arquitetura flexível, parametrizável e escalável, permitindo uma fácil e rápida modelagem. A plataforma pode ser descrita como uma arquitetura GALS (*Globally Asynchronous Locally Synchronous*) dividida em *clusters*. Os *clusters* são conectados através de uma rede assíncrona, chamada GANOC (*Globally Asynchronous Network-on-Chip*) [THO10].

A plataforma disponibiliza um modelo de programação para aplicações paralelas baseado em OpenCL [STO10] e NPM (*Native Programming Model*). O P2012 disponibiliza um framework para a implementação de componentes de aplicações e comunicação, podendo efetuar a simulação e captura de informações de aplicações em alto nível de abstração, dando suporte para modelos de programação, para análise, depuração e visualização de desempenho.

Ainda, no ponto de vista de software, a arquitetura de memória do P2012 é PGAS (*Partitioned Global Address Space*). Todos os processadores têm visibilidade completa de todas as memórias, sendo possível que todos processadores de um *cluster* carreguem e armazenem dados diretamente na memória L1 remota de outros *clusters*. O P2012 apresentado pelos Autores é implementado em tecnologia de 28nm, com quatro *clusters* homogêneos.

2.1.6 Busseuil et al.

Busseuil et al. [BUS11] descrevem o desenvolvimento de uma plataforma *open-source* escalável, chamada Open-Scale. A organização de memória da plataforma é de memórias locais distribuídas com comunicação baseada em troca de mensagens, sendo o NPU (*Network Processing Unit* [ALM09]) o principal componente do sistema. A NoC Hermes (mesma NoC utilizada na presente Tese) [MOR04] é utilizada para interconectar os elementos de processamento da plataforma.

O NPU é o principal componente do Open-Scale, sendo composto por: um processador SecretBlaze (MicroBlaze CPU) [BAR11], uma memória embarcada, um controlador de interrupções, uma UART (*Universal Asynchronous Receiver/Transmitter*), uma NI (*Network Interface*), um roteador e um barramento para interligar os componentes internos. A escalabilidade é feita pela replicação de NPUs.

A comunicação entre os elementos é feita através da API MPI. Mantendo a característica de memórias distribuídas, os Autores descrevem as aplicações baseado no modelo KPN (*Kahn Process Network*) [KAH77], ao qual possibilita a computação paralela de tarefas. O MPI disponibiliza um grande número de primitivas de computação para ambientes distribuídos. Alguns trabalhos especificam conjuntos de funções essenciais para troca de mensagens em sistemas embarcados. O Open-Scale, assim como outros trabalhos, utiliza os princípios básicos do MPI: *send* não bloqueante e *receive* bloqueante. Cada NPU executa um RTOS (*Real Time Operating System*) preemptivo.

Os Autores efetuam avaliações para demostrar a operação da plataforma Open-Scale.

Inicialmente é analisada a ocupação de memória, que após a compilação do sistema operacional a ocupação pode variar de 47KB até 57KB, dependendo das otimizações utilizadas pelo compilador. Outro ponto avaliado é o tempo de execução, o RTOS leva 237.000 ciclos de relógio para inicializar. Esse tempo compreende a inicialização de todos os serviços, incluindo a comunicação. A terceira e última avaliação é o desempenho da comunicação. Para enviar 200 bytes de um pacote, o Open-Scale consome aproximadamente 10.000 ciclos de relógio.

2.1.7 Paulin et al.

Paulin et al. [PAU06] apresentam um sistema chamado MultiFlex, que é uma ferramenta de mapeamento de aplicações para MPSoCs. O projeto MultiFlex disponibiliza a modelagem de plataformas através de encapsulamento e abstração de informações. Pode-se implementar SoCs para diversas finalidades com variações de graus de eficiência.

Os Autores desenvolveram o MPSoC StepNP [PAU04], assim como aplicações baseadas em troca de mensagens e multiprocessamento simétrico. A plataforma StepNP é composta de modelos de processadores reconfiguráveis, uma NoC para interconectar os componentes do sistema e periféricos. A plataforma inclui processadores RISC, tais como: MIPS (32-64bits), ARMv4 e PowerPC.

Dois modelos de programação são utilizados pelo MultiFlex: (*i*) DSOC (*Distributed System Object Component*) baseado no modelo de troca de mensagens; (*ii*) SMP (*Symmetrical Multiprocessing*) baseado em memória compartilhada. O modelo DSOC tem suporte à computação distribuída heterogênea, com características semelhantes ao CORBA [COR15] e Microsoft DCOM (*Distributed Component Object Model*) [DCO15], baseado em troca de mensagens. O SMP tem suporte a multithreading acessando memórias compartilhadas. Os conceitos de programação do SMP são similares ao Java embarcado e ao Microsoft C#.

Os Autores apresentam os resultados do mapeamento de uma aplicação de gerenciamento de tráfego de internet a 2.5Gb/s e uma aplicação de vídeo MPEG4 de resolução VGA a 30fps. O uso da ferramenta de mapeamento proposta, MultiFlex, disponibiliza uma rápida exploração de algoritmos descritos em SMP e DSOC, automaticamente mapeados em arquiteturas paralelas.

2.1.8 Meier et al.

Meier et al. [MEI10] apresentam um sistema que possibilita a prototipação rápida de MPSoCs baseado em uma abordagem orientada a modelos, chamado LavA. O modelo LavA tem por objetivo reduzir a complexidade de sistemas embarcados, abstraindo informações de *hardware* e outras configurações de baixo nível, disponibilizando ao projetista um modelo que descreve tais configurações na perspectiva de nível de sistema.

Para prover diferentes características de comportamento, como velocidade de computação, tamanho ou frequência, três tipos de processadores são avaliados. Cada processador pode ser

conectado com periféricos (UART e timer) via barramento *wishbone* [HER10]. A comunicação entre os processadores ocorre por mecanismos de troca de mensagens. A plataforma possibilita três diferentes tipos de interconectores: barramentos, *rings* e conexões ponto-a-ponto.

O fluxo de projeto do LavA é dividido em dois estágios: o primeiro estágio requer interversões diretas do projetista, sendo o segundo estágio um processo totalmente automático. No primeiro estágio o tempo de projeto dificilmente pode ser medido, porque depende da experiência do projetista e fatores como quantidade de periféricos conectados em cada processador e a complexidade das conexões entre os processadores. Para conseguir efetuar uma avaliação, os Autores definiram uma configuração base do SoC com três periféricos conectados a cada processador, sendo 8 o total de processadores. Usando esses parâmetros, o tempo de configuração manual do SoC leva em torno de 960 segundos. Os resultados do segundo estágio apresentaram um tempo em torno de 2 segundos para a geração automatizada.

2.1.9 Tian et al.

Tian et al. [TIA09] apresentam uma avaliação de desempenho de mecanismos de sincronização em MPSoCs. Para realizar tal avaliação, os Autores projetaram um MPSoC baseado em NoC com 16 elementos de processamento [FID08]. Cada PE do sistema integra um processador Xilinx MicroBlaze conectado a uma memória local de 64Kb. Todos os PEs são conectados a uma NoC de dados e outra NoC de sincronização, através de interfaces OCP-IP (*Open Core Protocol International Partnership*) [WAN08]. A NoC de sincronização conecta uma memória compartilhada, ao qual é efetuada a sincronização entre todos os PEs. A NoC de dados conecta quatro controladores DDR2 (*Double Data Rate*), que conectam quatro memórias DDR2 de 256Mb. O MPSoC descrito foi implementado em um FPGA Xilinx Virtex-4 FX140.

Segundo os Autores, mecanismos de sincronização devem seguir as seguintes características: baixa latência, pouco tráfego de mensagens na NoC, escalabilidade e baixo custo de armazenagem. São apresentadas duas formas de sincronização: *read-modify-write* e LL-SC (*Load-Linked Store-Conditional*). O LL-SC é um protocolo OCP (*Open Core Protocol*) que bloqueia a sincronização, sendo um conjunto atômico de transferências, sendo um mecanismo eficiente para gerenciar acessos exclusivos a recursos compartilhados.

Para os resultados, os Autores avaliam o LL-SC com um mecanismo de sincronização bloqueante, o qual bloqueia o processador até que ele encontre o que o deixou em espera. Os resultados mostraram que o LL-SC levou mais tempo do que o mecanismo de sincronização bloqueante, em todos os cinco cenários avaliados.

2.1.10 Discussão

A Tabela 1 compara os trabalhos apresentados previamente. A Tabela avalia cinco caraterísticas que estão diretamente ligadas aos objetivos da Tese. A primeira característica é a

descrição da arquitetura de comunicação. A maioria dos trabalhos, como [ZHA13] [LUK08] [BUS11] entre outros, utilizam NoC como infraestrutura de comunicação. Um ponto de destaque analisado nas plataformas é a utilização de níveis abstratos para modelagem de MPSoCs. Uma das contribuições desta Tese é a validação de uma plataforma, de sistema distribuído de memória, descrita em três diferentes níveis abstratos, além do desenvolvimento de um MPSoC baseado em NoC, com memória compartilhada, em alto nível de abstração. Nota-se que nos trabalhos analisados temos diversos níveis de abstração. Alguns Autores utilizam plataformas virtuais [REK13] [BES12] [ZHA13], enquanto outros descrevem plataformas em níveis mais baixos como RTL sintetizável [PAU06] e, em muitos casos, realizando a síntese para dispositivos FPGAs [LUK08] [BEN12] [BUS11] [MEI10] [TIA09]. Com o uso de plataformas virtuais, [ZHA13] e [LUK08] apresentam comparativos de desempenho entre plataformas de diferentes níveis de abstração. Os trabalhos analisados mostram uma variedade de organizações de memórias e sistemas operacionais, o que vem sendo uma área emergente na pesquisa de MPSoCs.

Tabela 1 - Comparativo entre os trabalhos de MPSoCs analisados.

Plataformas Multiprocessadas					
Ref.	**Arquitetura Comunicação**	**Modelagem**	**Hierarquia de Memória**	**Sistema Operacional**	**Comparativo entre níveis abstratos**
[REK13]	Barramento	OVP	Distribuída e Compartilhada	Linux embarcado	Sem comparações
[BES12]	Barramento	ArchC e SystemC TLM2.0	Compartilhada	Não relata uso	Sem comparações
[ZHA13]	NoC	OVP e SystemC TLM2.0	Distribuída e Compartilhada	Usa, não descreve qual	OVP/SystemC foi 40 vezes mais rápido que simulações em RTL
[LUK08]	NoC	PBD (*framework* EDK)	Distribuída e Compartilhada	Não relata uso	Testes com Xilinx EDK obtiveram os mesmos resultados sintetizados em um Virtex-II
[BEN12]	NoC (clusters) e barramento	PBD (*framework* EDK) – dispositivo Xilinx Zynq	Distribuída e Compartilhada	Linux embarcado	Sem comparações
[BUS11]	NoC	PBD (*framework* EDK) – dispositivo Virtex-V LX 110T	Distribuída	RTOS Preempetivo	Sem comparações
[PAU06]	NoC	RTL sintetizável	Distribuída e Compartilhada	Linux	Sem comparações
[MEI10]	Barramento	PBD (*framework* EDK) – dispositivo Virtex-V LX 110T	Compartilhada	Não relata uso	Sem comparações
[TIA09]	NoC	PBD (*framework* EDK) – dispositivo Xilinx-4 FX140	Distribuída e Compartilhada	Não relata uso	Sem comparações
Tese	**NoC**	**VHDL/RTL/OVP e ArchC**	**Distribuída Local / Compartilhada**	**Gerência por µkernel / Sem SO**	**Simulação OVP foi 5 vezes mais rápido que RTL-SystemC, que foi 155 vezes mais rápido que RTL-VHDL**

2.2 Métodos de Análise e Avaliação de Sistemas com Restrições de Tempo-Real

Nesta Seção apresenta-se o estado da arte relacionado ao estudo de plataformas com característica de tempo-real. Apresenta-se mecanismos de mapeamento e escalonamento voltados para MPSoCs que executam aplicações com restrições de tempo-real.

2.2.1 Indrusiak

Indrusiak [IND14] apresenta um método analítico para avaliar se um sistema embarcado multiprocessado baseado em NoC pode cumprir os requisitos de aplicações de tempo-real. Em aplicações críticas de segurança, com alto nível de restrições, um sistema embarcado deve reunir todos os requisitos necessários para atender a demanda imposta pelo comportamento desse tipo de aplicação.

Para avaliar se o MPSoC cumpre as exigências de aplicações de tempo-real, Indrusiak propõe o uso de dois métodos de análise: (*i*) RTA (*Response Time Analysis*) [AUD93]; (*ii*) análise de escalonabilidade de fluxo de tráfego na NoC [SHI10A]. O RTA é uma técnica para avaliar como a interferência de tarefas com alta prioridade pode afetar o tempo de execução das tarefas de baixa prioridade. A técnica RTA se torna a melhor escolha, pois a análise de escalonamento de fluxo de tráfego na NoC possui uma restrição de número limite de tarefas mapeadas em cada processador, não podendo avaliar processador executando múltiplas tarefas.

A plataforma desenvolvida para avaliação dos cenários de testes é um MPSoC 4x4 com 16 PEs interconectados por uma NoC. A NoC possui um roteamento não adaptativo e arbitragem com prioridades, tal como a NoC Hermes [MOR04] e QNoC (*Quality-of-Service NoC*) [BOL04]. A implementação mais comum de arbitragem de prioridade é baseada em canais virtuais (VCs) [BJE04], a qual possibilita que pacotes com alta prioridade interrompam a transmissão de pacotes de baixa prioridade.

Para validar os dois métodos apresentados, o Autor avaliou a plataforma com um benchmark baseado em algoritmos de veículos autônomos [SHI10B], composto por 39 tarefas comunicantes com funcionalidades como controle de navegação, controle de vibração e detecção de obstáculos. O período de injeção de dados das tarefas na NoC varia de 0,04 a 1 segundo, com volume de comunicação entre 1 e 76kB. Foram gerados três cenários com diferentes algoritmos de mapeamento de tarefas. Os cenários de testes foram simulados em 200 segundos, sendo um tempo aceitável para cobrir o tempo de vida de uma aplicação. Através dos métodos de análise pode-se avaliar diferentes algoritmos de mapeamento, verificando se as restrições de tempo-real são atendidas.

2.2.2 Zhou et al.

Zhou et al. [ZHO16] apresentam uma proposta de escalonamento estático para MPSoCs heterogêneos voltados para sistemas de tempo-real. A heurística de escalonamento apresentada

pelos Autores é dividida em dois estágios: (*i*) minimizar o consumo de energia dinâmica do sistema, distribuindo as tarefas com restrição de tempo entre vários processadores; (*ii*) diminuir o pico de temperatura de cada processador, utilizando o tempo de folga dos processadores. As tarefas modeladas são periódicas e as restrições de tempo são definas a partir de três parâmetros: (*i*) *deadline;* (*ii*) período; e (*iii*) *worst-case execution time* (WCET, pior tempo de execução).

Para alcançar os objetivos propostos, os Autores apresentam a técnica RTFA (*Real-Time Feasibility Analysis*) para analisar se os prazos de tempo das tarefas são atendidos. Além do RTFA, os Autores apresentam a técnica TFA (*Thermal Feasibility Analysis*) para verificar as restrições térmicas de determinada tarefa a ser escalonada no processador. Se o pico limite de temperatura do processador for violado, as tarefas que violaram as constantes térmicas são movidas para o conjunto de tarefas a serem realocadas.

As técnicas propostas são avaliadas comparando-as a outras três abordagens: (*i*) *Rate Monotonic First Fit* (RMFF); (*ii*) *Rate Monotonic Best Fit* (RMBF); e (*iii*) *Hybrid Worst-fit Generic Algorithm* (HWGA). Os algoritmos RMFF e RMBF atribuem prioridades às tarefas com base nos períodos das tarefas. Tarefas com períodos mais curtos têm maiores prioridades. Por outro lado, o HWGA atribui as tarefas com maiores prioridades aos processadores com capacidade máxima restante. Os resultados demonstram que as técnicas propostas não só podem estimar uma precisão de consumo de energia com erro relativo de 3,9%, como também apresentaram o *speedup* de 14,5 vezes em relação ao tempo total de execução. As avaliações mostraram que a abordagem proposta pode consumir aproximadamente 25% menos energia comparado aos demais algoritmos.

2.2.3 Chantem et al.

Chantem et al. [CHA08] apresentam uma formulação de programação linear de inteiros mistos (MILP, do inglês, *Mixed-Integer Linear Programming*) para atribuir e programar tarefas com restrições de tempo-real (*hard-RT*) para minimizar o pico de temperatura do MPSoC. A formulação considera variações térmicas temporais e espaciais. É proposta uma heurística de escalonamento de tarefas na qual o método de cálculo da temperatura do PE depende do tempo de duração da tarefa. Utilizou-se fórmulas de análise térmica para avaliar quando a duração da tarefa é relativamente maior do que o tempo que o PE poderia executar, antes de chegar ao pico de temperatura.

A análise térmica estima a transferência de calor através de materiais heterogêneos entre produtores de calor (por exemplo, transistores) e consumidores de calor (por exemplo, dissipadores de calor ligados a um MPSoC). No escalonamento de tarefas, os Autores adotaram um modelo de fluxo de calor com baixa granularidade, para equilibrar a eficiência e exatidão da análise térmica. A temperatura de cada elemento térmico pode ser expressa em função do seu consumo de energia, da temperatura do ambiente e das temperaturas dos elementos térmicos vizinhos. Os

resultados apresentaram benefícios significativos (melhoria de >30°C) de temperatura para MPSoCs com vários processadores, usando o modelo térmico proposto.

2.2.4 Bonilha et al.

Bonilha et al. [BON15] apresentam heurísticas de mapeamento de aplicações com restrições de tempo para arquiteturas multiprocessadas, com NoC como meio de comunicação, utilizando algoritmos genéticos. Os Autores utilizam o escalonador por prioridade fixa, executando tarefas periódicas. Três heurísticas de mapeamento são definidas para guiar os Autores na construção do algoritmo genético de mapeamento: (*i*) balanço de utilização entre os processadores, não sobrecarregando os processadores; (*ii*) conhecer a quantidade de prazos não atendidos, para saber se o mapeamento atende a todas as tarefas; e (*iii*) estressar a NoC, isso poderá afetar o comportamento temporal do sistema, gerando sobrecarga aos roteadores.

A ideia do uso de algoritmo genético é evoluir uma população de soluções para um determinado problema usando operadores inspirados na variação genética e seleção natural. O algoritmo genético usa uma analogia com a biologia evolutiva para se mover dentro do espaço de uma solução. A abordagem proposta pelos Autores pressupõe que cada solução no espaço de soluções pode conter vestígios da solução ideal e que a qualidade de determinada solução está diretamente relacionada com a proximidade da solução ideal. Considerando cada solução como um indivíduo, com determinada solução mapeada em seu código genético (cromossomo) e uma medida de quão próxima esta solução é a ideal (conhecido como *fitness*).

Para o mapeamento de tarefas com base em algoritmo genético, o cromossomo tem que ser modelado de forma que represente uma solução para o problema. O cromossomo escolhido pelos Autores é composto por uma matriz de pares (tarefa, processador), com uma posição fixa para cada tarefa no conjunto de tarefas. A ordenação da matriz é feita por tarefa, de modo que qualquer posição em determinado cromossomo representará a mesma tarefa em qualquer outro cromossomo.

Para avaliar o algoritmo genético proposto, usou-se um conjunto de 78 tarefas sintéticas sendo que 39 apresentam restrições de tempo-real e outras 39 são utilizadas para geração de fluxo de dados de comunicação da rede. Os resultados mostraram que a heurística proposta atende 100% dos requisitos de tempo de computação e 95% para os fluxos de comunicação.

2.2.5 Zaykov et al.

Zaykov et al. [ZAY13] apresentam um framework para contabilizar e distribuir a folga (conhecido como *slack time*, sendo o tempo restante entre o fim da execução de um *job* até o tempo de seu *deadline*) entre os processadores. A proposta é dividida em três quesitos: (*i*) os tempos de folga são distribuídos entre as tarefas, potencialmente mapeadas em diferentes processadores, através de sincronização entre tarefas; (*ii*) os tempos de folga podem ser

distribuídos em duas direções, do produtor para o consumidor ou vice-versa; e (*iii*) o tempo de folga pode ser estático ou dinâmico. O modelo de aplicação utilizado consiste em um conjunto de tarefas que se comunicam via *tokens* através de canais FIFO. Para avaliar a taxa de transferência das tarefas os Autores utilizam o MCM (*Maximum Cycle Mean*).

Os Autores modelam a folga estática como uma referência do tempo até que uma tarefa tenha que terminar sua próxima iteração. Esta definição não é a folga estática, mas sim as variáveis necessárias para calculá-la. A folga dinâmica é modelada como sendo a duração de tempo que a iteração atual necessita para terminar. Para efetuar estes cálculos, em cada processador é adicionado um bloco de *hardware* que calcula o tempo de folga durante a comunicação entre tarefas. Os resultados apresentam uma redução no consumo total de energia dos processadores em 27%, aumentando o tempo de execução das aplicações em até 4%.

2.2.6 Zhaoguo et al.

Zhaoguo et al. [ZHA09] apresentam uma otimização em tempo-real para vazão de energia e temperatura em MPSoCs. A principal contribuição é reduzir o consumo através de técnicas de mapemento de tarefas. A ideia da proposta é executar aplicações quando o processador estiver frio e a carga de trabalho estiver alta, e colocar o processador em modo *hold* quando ele estiver quente ou quando a carga de trabalho estiver baixa.

O sistema pode apresentar dois estados: modo ativo e modo ocioso. Durante o modo ativo as tarefas são executadas e o sistema consome energia. Durante o modo ocioso as tarefas são bloqueadas e o sistema passa a consumir pouca energia. Para desempenhar estas definições, os Autores apresentam o algoritmo TALO (*Temperature-Aware Leakage Optimization*). Tal algoritmo atribui alta prioridade aos processadores mais frios do sistema.

Os resultados mostram um comparativo do TALO com DVFS (*Dynamic Voltage Frequency Scale*) e TALK (*Thermal-Aware Leakage*). Os experimentos demonstraram que o TALO é mais eficiente em consumo de energia. Ele pode reduzir a o consumo energético em até 70%, comparado com o DVFS. Também, podendo diminuir a temperatura dos processadores do sistema. Por outro lado, o TALO apresenta os mesmos ganhos energéticos proposto pelo TALK. Porém, o TALK é um mecanismo para qualquer tipo de sistema (não garantindo atendimento aos requisitos de tempo), enquanto o TALO pode ser usado para sistemas RT.

2.2.7 Amin et al.

Amin et al. [AMI12] propõem uma abordagem em nível de sistema para tolerância a falhas em sistemas multiprocessados que podem ser customizados de acordo com os requisitos e restrições das aplicações. A metodologia proposta é feita pelo balanceamento dos PEs críticos (FT, tolerante a falhas) e não críticos (HP, de alto desempenho) em um MPSoC baseado em NoC. Os processadores FT tem recuperação rápida de erro, mas uma área maior em comparação ao HP. O

FT, também, apresenta uma execução mais lenta devido a sobrecargas de tempo causados por circuitos complexos de detecção e correção de erros.

A proposta trabalha com três métodos de injeções de falhas: (*i*) sem falha; (*ii*) baixa taxa de injeção de falhas; e (*iii*) alta taxa de injeção de falhas. A latência de comunicação da plataforma depende não apenas do tamanho da mensagem, mas também do mapeamento de recursos. Isso ocorre, pois os fluxos simultâneos de dados compartilham os mesmos recursos de rede, podendo causar conflitos de rede e aumentar o atraso médio no envio da mensagem.

Os testes são executados em um MPSoC de instância 10x10, executando um benchmark com 2000 tarefas sintéticas com alto poder de comunicação. Um gerador de falhas externo é responsável por injetar falhas aleatoriamente nos processadores, independente da área e da unidade. Desta maneira, tanto os processadores HPs quanto os FTs estão suscetíveis a serem defeituosos. Se uma falha for detectada em tempo de execução, a tarefa falha será reexecutada utilizando um tempo de folga pré-definido pelo escalonador de tempo-real.

2.2.8 Discussão

A Tabela 2 compara os trabalhos apresentados previamente, onde são apresentadas pesquisas que utilizam diferentes métodos analíticos como função custo de mecanismos em plataformas multiprocessadas com restrições de tempo-real, como mapeamento e escalonamento.

Tabela 2 - Comparativo entre os trabalhos de métodos analíticos para sistemas RT.

Métodos Analíticos para Sistemas RT				
Ref.	**Arquitetura Comunicação**	**Tipo de Aplicação**	**Método Analítico**	**Mecanismo Alvo**
[IND14]	NoC	Hard RT	RTA	Mapeamento
[ZHO16]	Barramento	Soft RT	RTFA	Escalonamento
[CHA08]	Barramento	Hard RT	Análise térmica	Escalonamento
[BON15]	NoC	Hard RT	RTA	Mapeamento
[ZAY13]	NoC	Soft RT	MCM	Escalonamento
[ZHA09]	Barramento	Soft RT	TALO	Escalonamento
[AMI12]	NoC	Hard RT	RTA	Mapeamento
Tese	**NoC**	**Hard RT Soft RT**	**RTA**	**Escalonamento e Mapeamento**

A Tabela avalia quatro caraterísticas: (*i*) arquitetura de comunicação; (*ii*) tipo de aplicação de tempo-real; (*iii*) método analítico; e (*iv*) mecanismo alvo. A primeira característica, assim como na Tabela 1, é a descrição da arquitetura de comunicação, os trabalhos são distribuídos entre infraestrutura de comunicação NoC e barramentos. Como mencionado anteriormente, sistemas com restrições de tempo-real são classificados em Hard RT (como podemos notar em [CHA08]

[BON15] [AMI12]) e Soft RT (como podemos notar em [ZHO16] [ZAY13] [ZHA09]). Outra característica analisada nos trabalhos é o uso de métodos analíticos como função custo dos mecanismos alvos (por exemplo, mapeamento e escalonamento). Trabalhos como [ZHO16] [CHA08] [ZHA09] utilizam método de análise para controle térmico. Já trabalhos como [BON15] [AMI12] [IND14] utilizam o método analítico RTA para verificar se o sistema atende às restrições de tempo-real. Os trabalhos analisados apresentam resultados com base, principalmente, em aplicações com restrições de tempo-real.

2.3 Benchmarks

Nesta Seção apresenta-se benchmarks utilizados para os testes dos mecanismos propostos nesta Tese.

2.3.1 Woo et al.

Woo et al. [WOO95] apresentam e caracterizam os programas desenvolvidos no SPLASH-2 (*Stanford ParalleL Applications for SHared memory*). O principal objetivo do trabalho é caracterizar os programas do SPLASH-2 em termos de propriedades básicas e arquitetural, informações importantes na definição dos parâmetros para avaliação de sistemas.

O benchmark SPLASH-2 consiste em uma mistura de oito aplicações e quatro *kernels*, que representam uma variedade de algoritmos de programação científica, de engenharia e gráficos computacionais. Na presente Tese são adaptadas três aplicações caracterizadas e descritas no SPLASH-2: FFT (*Fast Fourier Transform*), LU (Lower and Upper) e Water-Spatial. Os autores descrevem que a aplicação FFT é uma versão 1-D complexa com a comunicação entre tarefas minimizada, visando uma maior computação e não comunicação. A aplicação LU trabalha com fatoração de matrizes. Já a aplicação *Water-Spatial* avalia a força e o potencial hídrico que ocorrem ao longo do tempo em um sistema de moléculas de água. Essas avaliações são computadas por um algoritmo de complexidade $O(n^2)$, e um método preditor-corretor é usado para integrar movimento das moléculas ao longo de determinado período.

2.3.2 Guthaus et al.

Guthaus el al. [GUT01] apresentam um benchmark com 35 aplicações, chamado MiBench. O benchmark é composto por seis categorias, incluindo: controle automotivo e industrial, redes, segurança, dispositivos eletrônicos, automação residencial e telecomunicações. Na presente Tese são adaptadas seis aplicações do MiBench: Basicmath, Susan, Dijkstra, Sha (*Security Hash Algorithm*), StringSearch e FFT. As aplicações descritas no MiBench são aplicações que executam em sistemas com apenas um processador. As aplicações especificadas acima, que são utilizadas na Tese, serão mais bem detalhadas na Seção 4.1.

2.3.3 Iqbal et al.

Iqbal et al. [IQB10] propõem um benchmark paralelo *open-source* para sistemas embarcados, chamado ParMiBench. Sete são as aplicações do MiBench [GUT01] presentes no benchmark proposto pelos Autores. Tais aplicações foram convertidas para executarem em sistemas multiprocessados. As implementações paralelas implementam funções da API PThread e são descritas em linguagem C. Na presente Tese são adaptadas cinco aplicações do ParMiBench: Dijkstra, Susan, BasicMath, Sha e StringSearch. De acordo com os Autores, o MiBench é um benchmark altamente utilizado para sistemas embarcados uniprocessados. A diferença entre as aplicações do ParMiBench comparadas com as aplicações do MiBench é que elas executam em sistemas embarcados multiprocessados.

Os Autores descrevem que a aplicação Susan é uma aplicação para reconhecimento de imagem, ao qual reconhece bordas e cantos de imagens de ressonância magnética do cérebro humano. A aplicação BasicMath é usada para cálculos matemáticos em processadores embarcados, tais como: resolução de funções cúbicas, conversão de ângulos de graus para radianos e raiz quadrada. O Dijkstra calcula o menor caminho entre dois pontos representados por um grafo em matrizes adjacentes. A aplicação StringSearch encontra palavras específicas em um texto. O algoritmo SHA é utilizado para criptografia de dados.

2.3.4 Discussão

A Tabela 3 apresenta um comparativo de trabalhos propostos na área de *benchmark suite*. São apresentados os trabalhos que serviram como base ao MP2I-BencH, apresentado no próximo Capítulo. [WOO95] e [GUT01] apresentam benchmarks para propósito geral, como segurança, telecomunicações e veículos autônomos. [IQB10] apresentam um benchmark voltado para cálculos matemáticos, com alto poder de computação. Todos os benchmarks analisados são descritos para execução em sistemas com memórias compartilhadas. Com isso, a outra contribuição da presente Tese é disponibilizar um *benchmark suite* para MPSoCs baseados em sistema de memórias distribuídas.

Tabela 3 - Comparativo entre os trabalhos de Benchmarks analisados.

Benchmarks				
Ref.	**Tipo Processamento**	**Número de Aplicações**	**Aplicação Foco**	**APIs de Comunicação**
[WOO95]	Multiprocessado	8 aplicações e 4 *kernels*	Propósito geral	PThread
[GUT01]	Uniprocessado	35 aplicações	Propósito geral	PThread
[IQB10]	Multiprocessado	7 aplicações	Cálculos matemáticos	PThread
Tese	**Multiprocessado**	**9 aplicações**	**Propósito geral**	**MPI (memória distribuída) e PThread (memória compartilhada)**

3 MODELAGEM DE PLATAFORMAS

Neste Capítulo apresentam-se a primeira contribuição da Tese: o desenvolvimento de plataformas virtuais para modelagem de MPSoCs. Nesse sentido, a plataforma HeMPS [WOS07] foi modelada utilizando: VHDL, nível de abstração RTL; SystemC, nível de abstração RTL; e OVP, nível de abstração *untimed*. Cada modelo contém seu próprio sistema operacional (*µkernel*). A HeMPS utiliza dois *µkernels* distintos: (*i*) mestre, responsável pela gerência de cluster (uma região do MPSoC contendo uma quantidade parametrizável de processadores) e do sistema como um todo; (*ii*) *slave*, responsável pela execução multitarefa. Para validar a plataforma, com os três modelos, foi necessário descrever de forma unificada os diferentes *µkernels*. Para isso, foi necessário o desenvolvimento de uma camada em software para abstrair a modelagem do *hardware*, denominada HAL (*Hardware Abstraction Layer*).

Buscando determinar as limitações de arquiteturas multiprocessadas, observou-se a necessidade de desenvolver plataformas com diferentes hierarquias de memória, sendo esta a segunda contribuição desta Tese. A HeMPS é uma plataforma com memória localmente distribuída entre os PEs e com gerência distribuída de recursos. Como visto no Capítulo do Estado da Arte, diversos benchmarks foram desenvolvidos para serem executados em plataformas com arquitetura de memória compartilhada. Desta forma, desenvolveu-se uma plataforma (utilizando ArchC) com tal característica. A plataforma com arquitetura de memória compartilhada foi inserida ao MPSoCBench [DUE14], adicionando a implementação e adaptação da NoC Hermes/SystemC TLM2.0 [MOR04] à arquitetura. Para avaliar a escalabilidade de MPSoCs com diferentes organizações de memórias, desenvolveu-se uma terceira plataforma, descrita em OVP, com organização de memória similar à HeMPS, mas sem gerência de recursos distribuída. As ADLs utilizadas para modelagem das plataformas são apresentadas na Seção 3.1. As implementações destas plataformas modeladas são apresentadas na Seção 3.2.

3.1 Plataformas Virtuais

O uso de Plataformas Virtuais (VPs – *Virtual Platforms*) para desenvolvimento de *software* embarcado proporciona benefícios, tais como: menor custo de desenvolvimento, aumento da qualidade do *software* (pois este tipo de plataforma tem alto poder de teste) e redução de riscos envolvidos com o desenvolvimento de *software*. VPs são meios eficientes no qual as funcionalidades do *software* e do *hardware* desejado podem ser projetadas e verificadas em conjunto nas fases iniciais do projeto [REK13]. Dessa forma, a integração do *hardware* com o *software* é feita desde o início do processo de implementação, ao contrário do desenvolvimento tradicional onde o *software* só é implementado após o *hardware* estar finalizado.

Esta Seção apresenta a modelagem de diferentes configurações de MPSoCs com

plataformas virtuais. A simulação de uma plataforma virtual permite verificar o comportamento do sistema completo, com os componentes de *hardware* (como CPU, memórias e periféricos), e o *software* (sistema operacional e aplicações do usuário). Visando acelerar o tempo de desenvolvimento de *software*, alguns ambientes de desenvolvimento de VPs oferecem conjuntos de modelos de processadores e sistemas de memória, permitindo a análise de diferentes aplicações ou sistemas operacionais em arquiteturas multiprocessadas. Exemplos destes ambientes incluem OVPsim [OVP15], ArchC [RIG03], GEM5 [BIN11], Simics [SIM15], SimpleScalar [AUS02].

As plataformas de alto nível de abstração desenvolvidas na presente Tese são simuladas utilizando OVPsim e ArchC. As próximas Subseções apresentam os ambientes de desenvolvimento de plataformas virtuais utilizados nesta Tese. A Subseção 3.1.1 apresenta o OVP, seus modelos arquiteturais, APIs de desenvolvimento e simulador. A Subseção 3.1.2 apresenta o ArchC, suas definições de arquitetura e o padrão de comunicação.

3.1.1 Open Virtual Platform (OVP)

OVP [OVP15] é um ambiente de desenvolvimento de plataformas virtuais, proposta pela Imperas, que possibilita a modelagem e a simulação de sistemas embarcados. OVP iniciou em 2008 como uma ferramenta de código aberto, flexível e gratuita. Hoje, as ferramentas mais avançadas do ambiente OVP requerem licença paga (como o simulador de MPSoCs heterogêneos, o *OVPmanager*). Com as ferramentas providas pela plataforma é possível descrever arquiteturas homogêneas e heterogêneas, com hierarquias distintas de memória.

A plataforma é composta por uma API que possibilita a criação de modelos arquiteturais através da linguagem de programação C. Além da API, o OVP possui modelos de processadores (por exemplo: ARC600, ARM Cortex-A, Xilinx MicroBlaze, MIPS32, PowerPC), periféricos (por exemplo: FIFO, DMA e UART) e componentes de sistema (por exemplo: memórias caches e RAM).

Para a modelagem das plataformas, o ambiente OVP é composto por três componentes:

- ***OVPmodels***. O conjunto de modelos OVP consiste em um grupo de modelos (processadores e periféricos) que podem ser personalizados pelos projetistas conforme a necessidade do projeto. Além do modelo para simulação, o OVP disponibiliza o código fonte para que os projetistas possam adaptar os modelos de periféricos e processadores às suas arquiteturas
- ***OVP APIs***. As APIs disponibilizadas possibilitam que o usuário descreva o comportamento de processadores e periféricos para gerar plataformas virtuais. As APIs são escritas em linguagem C/C++ e dividas em cinco tipos com funcionalidades distintas: OP (*Open Platform*), ICM (*Innovative CPU Manager*), VMI (*Virtual Machine Interface*), BHM (*Behavior Models*) e PPM (*Peripherals Model*)
- ***OVPsim***. O OVPsim é a ferramenta utilizada para gerar os simuladores das plataformas

descritas com as APIs.

Os periféricos e modelos são interconectados na plataforma através das APIs ICM (*Innovative CPU Manager*) e OP (*Open Platforms*), que também é responsável por instanciar todos os componentes do sistema. Os comportamentos de cada modelo de processador são descritos utilizando a API VMI (*Virtual Machine Interface*). A VMI traduz as instruções que serão simuladas para executar no computador hospedeiro, sendo que a execução de múltiplos processadores é sequencial. Os periféricos são descritos utilizando a API PPM (*Peripherals Model*) e seus comportamentos são descritos pela API BHM (*Behavior Models*).

Quando o OVPsim executa uma plataforma multiprocessada, cada processador executa um número de instruções pré-definido (*quantum*) [OVP15]. O simulador, por padrão, define que um *quantum* executa 100.000 instruções por processador. A quantidade de milhões de instruções executadas por segundo (MIPS) deve ser definida na modelagem do processador. Os modelos de processadores fornecidos pelo OVP executam, por padrão, 100 MIPS.

Para exemplificar a modelagem de plataformas utilizando OVP, a Figura 2 representa um sistema com dois processadores, conectados a memórias locais, e uma memória compartilhada para comunicação. Todos os elementos do sistema (memórias scratchpad), memória compartilhada, processadores e barramentos) são componentes disponibilizados pela Imperas e instanciados através da API ICM.

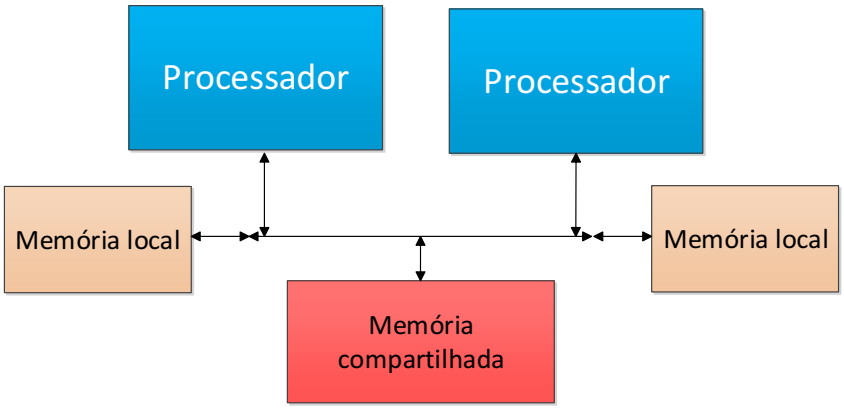


Figura 2 - Exemplo de modelagem de plataforma utilizando a ADL OVP.

O código apresentado na Figura 3 implementa a modelagem da plataforma apresentada na Figura 2. As linhas 3-7 instanciam novos processadores através da API ICM. Nas linhas 9-11 são criados os barramentos responsáveis pela interconexão dos componentes do sistema. As linhas 13-15 conectam os processadores aos barramentos. As memórias (*scratchpads* e compartilhada) são criadas nas linhas 17-20 e conectadas a seus devidos barramentos nas linhas 22-26. As linhas 28-30 carregam os códigos objetos (programa executável em formato ELF, *Executable and Linking Format*) na memória dos processadores. Por fim, na linha 33 invoca-se o método de inicialização da simulação da plataforma.

```
int main(int argc, char ** argv) {

    // cria o primeiro processador
    icmProcessorP processor0 = icmNewProcessor("cpu0", "or1k");

    // cria o segundo processador
    icmProcessorP processor1 = icmNewProcessor("cpu1", "or1k");

    // cria os barramentos dos processadores
    icmBusP bus1 = icmNewBus("bus1", 32);
    icmBusP bus2 = icmNewBus("bus2", 32);

    // conecta os processadores aos barramentos
    icmConnectProcessorBusses(processor0, bus1);
    icmConnectProcessorBusses(processor1, bus2);

    // cria as memórias definindo os seus tamanhos
    icmMemoryP shared = icmNewMemory("shared", 0x00010000);
    icmMemoryP local1 = icmNewMemory("local1", 0x00010000);
    icmMemoryP local2 = icmNewMemory("local2", 0x00010000);

    // conecta as memórias aos barramentos
    icmConnectMemoryToBus(bus1, shared);
    icmConnectMemoryToBus(bus2, shared);
    icmConnectMemoryToBus(bus1, local1);
    icmConnectMemoryToBus(bus2, local2);

    // carrega o código objeto na memória do processador
    icmLoadProcessorMemory(processor0, argv[1]);
    icmLoadProcessorMemory(processor1, argv[2]);

    // inicia a simulação da plataforma
    icmSimulatePlatform();

}
```

Figura 3 - Exemplo de código fonte para modelagem de plataforma utilizando a ADL OVP.

3.1.2 ArchC

ArchC [RIG03] é uma ADL baseada na linguagem SystemC, com recursos que permitem ao projetista explorar, testar e verificar arquiteturas de processadores através do simulador SystemC. O ArchC foi projetado pelo Laboratório de Sistemas Computacionais (LSC) do Instituto de Computação da Universidade de Campinas (IC-Unicamp).

SystemC é uma coleção de classes e *templates* C++ que disponibilizam mecanismos para modelar arquiteturas com comportamento temporal, concorrente e reativo, gerando uma especificação executável de um sistema. Com a linguagem ArchC é possível descrever hierarquias de memórias e arquiteturas de processadores. A modelagem de uma arquitetura é dividida em duas partes: (*i*) definição do conjunto de instruções da arquitetura, AC_ISA; (*ii*) modelagem dos elementos da arquitetura, AC_ARCH.

Na definição dos conjuntos de instruções de uma arquitetura (AC_ISA), o projetista descreve os detalhes relativos aos formatos de instruções, tamanhos e seus nomes. Nessa etapa do projeto deve-se descrever o comportamento de cada instrução e todas as demais informações

necessárias para decodificá-las. Na modelagem dos elementos da arquitetura (AC_ARCH) o projetista deve informar os dispositivos de armazenamento e estruturas de *pipeline*. No AC_ARCH podem-se descrever as portas pelo qual o processador irá se comunicar com componentes externos (como memórias compartilhadas, barramentos e memórias cache). Tal comunicação é efetuada através de modelagem TLM.

TLM (*Transaction Level Model*) é um padrão de modelagem, em alto nível de abstração, desenvolvido pela OSCI e adicionado à linguagem SystemC [GHE06]. Neste nível de abstração, os detalhes da comunicação entre os módulos modelados são separados da implementação das unidades funcionais. Segundo Martha [MAR06], a principal característica do nível TLM é a abstração da comunicação em termos de interfaces que implementam o conjunto de métodos. TLM modela um sistema como uma série de transações de escrita e leitura, simplificando os esforços de modelagem e diminuindo o tempo de simulação [DAN06].

Com base na AC_ISA e AC_ARCH, ArchC gera automaticamente ferramentas para geração e inspeção de código e, também, modelos de processadores executáveis para a representação de plataformas. É utilizada a extensão PowerSC [KLE07] do SystemC para coletar dados e processar a execução da simulação [MAD16A]. A conexão dos modelos do ArchC com a extensão PowerSC é feita através da biblioteca acPower que habilita a análise de potência dos módulos de processadores ArchC. O ambiente do ArchC disponibiliza modelos de processamento para utilização nas plataformas, tais como: SPARCV8, MIPS, ARM e PowerPC.

Para exemplificar a modelagem de plataformas utilizando a ADL ArchC, a Figura 4 representa um MPSoC com quatro processadores, um *locker* para controle da atomicidade dos dados e uma memória compartilhada para troca de comunicação ente os componentes do sistema. Os elementos são conectados através de uma NoC. Todos os elementos do sistema (memória compartilhada, NoC, processadores e *locker*) são componentes disponibilizados na biblioteca ArchC. Quando os processadores são instanciados, além do processador, o ArchC modela a cache de dados implicitamente.

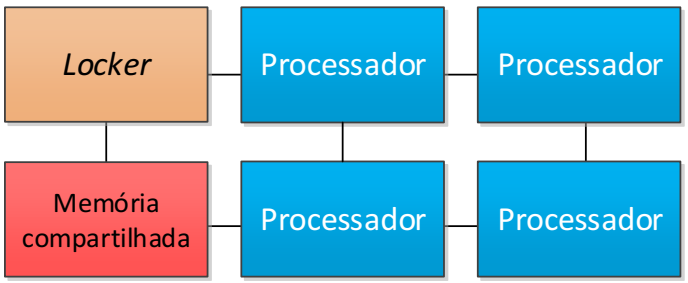


Figura 4 - Exemplo de implementação de um MPSoC 3x2 modelado com componentes ArchC.

O código apresentado na Figura 5 implementa a modelagem da plataforma apresentada na Figura 4, instanciando e conectando os componentes ArchC. A linha 3 instancia uma NoC já pré-configurada (conforme parâmetros informados na execução da plataforma). Nas linhas 5-9 são

criados e instanciados os processadores modelados em ArchC. As linhas 11-15 representam a criação dos *wrappers* que, nas linhas 17-20, conectam os processadores aos roteadores da NoC. As linhas 22-25 criam o *locker* e fazem a sua conexão ao roteador 01 da NoC. As linhas 27-30 criam a memória compartilhada e fazem a sua conexão ao roteador 00 da rede. Nas linhas 32-36 o ELF de inicialização da plataforma é carregado nos elementos de processamento. Na linha 39 inicia-se a simulação da plataforma.

```
1  int main(int argc, char ** argv) {
2      // instancia uma noc
3      noc noc("noc");
4
5      // cria os processadores
6      PROCESSOR_NAME proc0("proc0");
7      PROCESSOR_NAME proc1("proc1");
8      PROCESSOR_NAME proc2("proc2");
9      PROCESSOR_NAME proc3("proc3");
10
11     // cria os wrappers de conexão noc<->proc
12     ac_noc w_proc_01("w_proc_01", 0x01);
13     ac_noc w_proc_02("w_proc_02", 0x02);
14     ac_noc w_proc_11("w_proc_11", 0x11);
15     ac_noc w_proc_12("w_proc_12", 0x12);
16
17     proc0.MEM_port(w_proc_01.target_export);
18     proc1.MEM_port(w_proc_02.target_export);
19     proc2.MEM_port(w_proc_11.target_export);
20     proc3.MEM_port(w_proc_12.target_export);
21
22     // cria o locker
23     ac_noc_s w_locker_10("w_locker_10", 0x10);
24     w_locker_10.inPort(iR10out);
25     w_locker_10.outPort(iR10in);
26
27     // cria a memória
28     nocmem_nb w_mem_00("w_mem_00", 0x00);
29     w_mem_00.inPort(iR00out);
30     w_mem_00.outPort(iR00in);
31
32     // carrega o ELF de inicialização da plataforma nos processadores
33     load_elf(proc0, w_mem_00, arguments[0][1], MEM_SIZE);
34     load_elf(proc1, w_mem_00, arguments[1][1], MEM_SIZE);
35     load_elf(proc2, w_mem_00, arguments[2][1], MEM_SIZE);
36     load_elf(proc3, w_mem_00, arguments[3][1], MEM_SIZE);
37
38     // inicia a simulação da plataforma
39     sc_start();
40 }
```

Figura 5 - Exemplo de código fonte para modelagem de sistemas utilizando a ADL ArchC.

3.2 Plataformas multiprocessadas

Segundo Matilainen et al. [MAT11], aplicações embarcadas modernas estão se tornando cada vez mais complexas, e a arquitetura de memória define o modelo de programação adotado pelo MPSoC. Por exemplo, aplicações baseadas em *threads* são adequadas para arquitetura de

memória compartilhada, enquanto aplicações com a comunicação baseada em troca de mensagens são adequadas para arquitetura de memórias distribuídas. A organização de memória em MPSoCs representa um desafio para projetistas de *software* e *hardware*. A comunicação entre os processadores não é só uma função da infraestrutura de comunicação, mas também é a escolha correta da arquitetura de memória adotada para a plataforma.

Nesta Seção apresenta-se a modelagem de plataformas multiprocessadas com a utilização de diferentes ADLs e simuladores. A Seção 3.2.1 apresenta uma plataforma com arquitetura de memórias distribuídas e com gerência distribuída de recursos, modelada de três formas distintas: RTL-VHDL, RTL-SystemC e OVP. Na Seção 3.2.2 apresenta-se uma plataforma com arquitetura memória compartilhada descrita em alto nível de abstração, sem gerência de recursos. Também, na Seção 3.2.2, apresenta-se uma plataforma com arquitetura de memórias distribuídas, descrita em alto nível de abstração, sem gerência de recursos.

3.2.1 Plataforma com Arquitetura de Memórias Distribuídas

As modelagens apresentadas nesta Seção não contêm memórias *caches* e memórias compartilhadas. Os dados e os códigos objetos de aplicações são armazenados em memórias locais. A utilização de memória compartilhada induz alto tráfego na rede, gerando congestionamento na região da memória compartilhada. A comunicação entre os elementos do sistema com memórias distribuídas ocorre através de troca de mensagem, diminuindo a ocorrência de congestionamento e aumentando a eficiência energética no meio de comunicação [MAD16A].

Para a modelagem do *hardware* das plataformas, seja com memórias distribuídas ou memórias compartilhadas, necessita-se de simuladores com precisão de ciclo de relógio ou precisão aproximada de ciclo de relógio (*quasi-cycle accurate*) [BIN11]. Por outro lado, o desenvolvimento de *software* requer alta velocidade de simulação (como 100 MIPS, proporcionado pelo OVP [OVP15]) [LEM12]. Com tais requisitos conflitantes, é difícil cobrir todas necessidades de modelagem e simulação de plataformas em um único simulador. Além disso, para assegurar a correta funcionalidade de sistemas embarcados complexos, restrições de projetos como área, consumo energético e temperatura devem ser consideradas.

Levando em consideração tais restrições, esta Tese apresenta uma plataforma multi-nível, denominada HeMPS-ML, a qual combina diferentes técnicas de modelagem e simuladores. Alta flexibilidade na modelagem, depuração e simulação rápida são alcançadas com uso de plataformas abstratas modeladas com OVP. Plataformas com precisão de ciclo de relógio são modeladas com SystemC e RTL, proporcionando a geração de métricas precisas como área e consumo de energia.

A interoperabilidade entre os três modelos é garantida através de uma HAL e uma descrição unificada do *software* (tal como, sistemas operacionais, aplicações e modelos de comunicação). Nesta direção, o *software* pode ser modificado e executado em uma plataforma

descrita em alto nível (OVP), para validação de suas funcionalidades. O mesmo *software* pode ser executado em um modelo com velocidade intermediária de simulação, mas com precisão de ciclo de relógio (SystemC), ao qual pode-se buscar estimativas de desempenho, como tempo de execução de aplicações e energia consumida. Finalmente, através de um modelo de mais baixo nível (RTL VHDL) pode-se coletar informações mais precisas, como energia e área. As métricas obtidas através do modelo RTL (VHDL) são utilizadas nos modelos mais abstratos, permitindo que os mesmos possam estimar, por exemplo, energia e tempo de execução.

As três plataformas modeladas executam o mesmo *software* e contêm as mesmas características, tal como: (*i*) utilização da NoC Hermes [MOR04], devido a escalabilidade e paralelismo na comunicação; (*ii*) todos os elementos de processamento (PE) contêm um processador MIPS (Plasma-IP) conectado a uma memória local, um módulo DMA, uma NI (*Network Interface*) e um roteador conectado à NoC; (*iii*) o modelo de comunicação é baseado em troca de mensagens (MPI).

O MPSoC é dividido em *n* clusters e possui uma gerência de recursos distribuída [CAS13]. A Figura 6 apresenta uma visão simplificada da arquitetura adotada pelas diferentes modelagens. Na Figura, apresenta-se um MPSoC com tamanho 6x6, dividido em 4 clusters com dimensão 3x3.

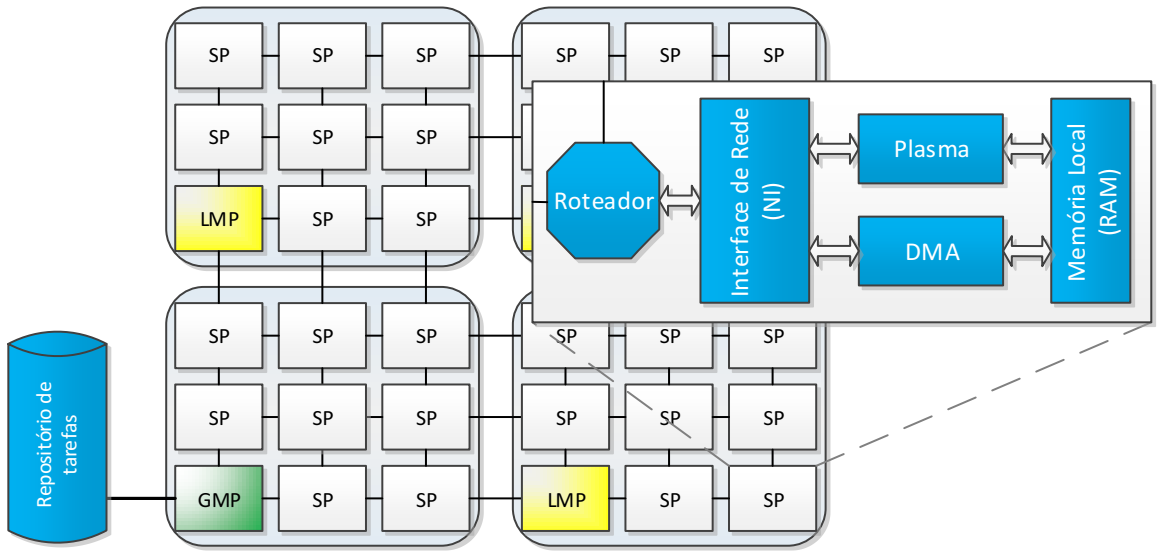


Figura 6 - Arquitetura de um MPSoC baseado em NoC, com gerência distribuída de recursos. GMP: *Global Manager* PE; LMP: *Local Manager* PE; SP: *Slave* PE. Fonte: [CAS13]

Todos os PEs possuem o mesmo *hardware*. Os PEs podem assumir diferentes funções, definidas por *software*: GMP (*Global Manager Processor*), LMP (*Local Manager Processor*) e SP (*Slave Processor*). O LMP é responsável pela gerência do cluster, como execução de funções de monitoramento e mapeamento de tarefas. O GMP é responsável pela gerência geral do sistema, pela comunicação com uma memória externa, chamada de repositório de tarefas, o qual contém as aplicações a serem executadas no sistema e contém todas as funcionalidades do LMP. Os SPs são responsáveis pela execução das tarefas das aplicações. Cada SP executa um sistema

operacional embarcado, que possibilita a comunicação entre os PEs e a execução de múltiplas tarefas.

A Seção 3.2.1.1 apresenta os modelos de hardware usados para descrever o MPSoC, e a Seção 3.2.1.2 o modelo de software unificado.

3.2.1.1 Modelagens de hardware em diferentes níveis de abstração

Para avaliar diferentes aspectos de projeto, três modelos do MPSoC são desenvolvidos em níveis de abstrações diferentes. A Figura 7 apresenta as principais vantagens e desvantagens da utilização de cada modelo. O modelo RTL-VHDL permite a simulação do MPSoC com precisão de ciclo de relógio e pode reportar estimativas de desempenho, potência e área. O modelo RTL-SystemC, também disponibiliza simulação com precisão de ciclo de relógio sendo descrito a partir do modelo RTL-VHDL, provê reduzido tempo de simulação comparado ao modelo RTL-VHDL. A implementação OVP sacrifica a precisão temporal para prover melhor depuração do *software*, simulando mais rapidamente do que os modelos anteriores. Uma característica muito importante do OVP é a flexibilidade da modelagem. OVP permite explorar diferentes módulos de *hardware*, tal como processadores, periféricos e memórias, sendo assim uma ferramenta de apoio para projetistas na tomada de decisões de projeto.

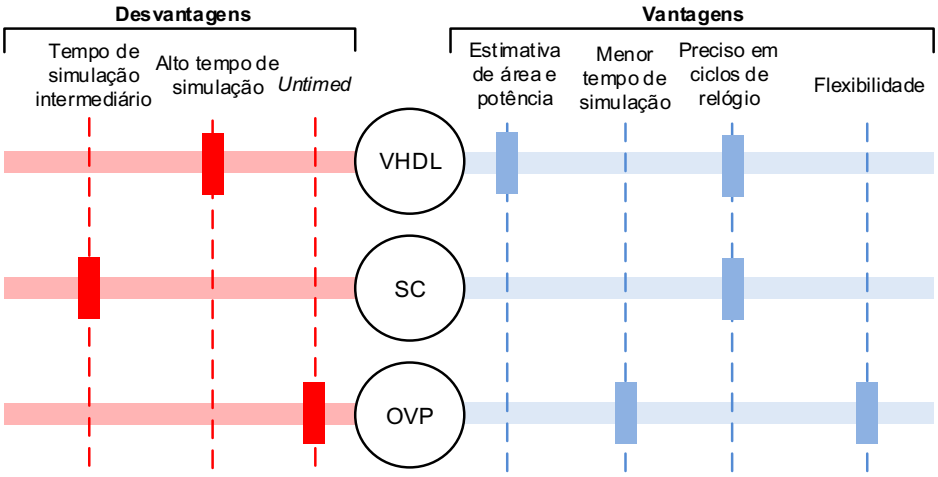


Figura 7 - Vantagens e desvantagens de cada modelo. Fonte: [MAD15]

A. Modelagem em VHDL

Modelagens de plataformas em nível de abstração RTL (*Register Transfer Level*) garantem precisão de ciclo de relógio. Abstrações RTL são utilizadas em linguagens de descrição de hardware para representar um circuito em baixo nível, tal como a linguagem VHDL. Além da precisão de ciclo de relógio, esta abstração permite gerar estimativas de área e potência através de ferramentas de síntese lógica e física.

Na modelagem VHDL o projetista pode parametrizar diversas características arquiteturais do MPSoC. A memória local pode ser configurada para ASIC (*Application-Specific Integrated Circuit*) ou FPGA (Xilinx Block RAMs). Em nível de rede é possível selecionar a profundidade do

buffer, algoritmo de roteamento, política de arbitragem, entre outros parâmetros.

O modelo RTL-VHDL tem como principal vantagem o fato de ser sintetizável, já havendo sido realizado o desenvolvimento de uma versão do sistema com 9 PEs em um dispositivo FPGA. O protótipo no FPGA contém além do MPSoC três módulos adicionais: (*i*) interface de comunicação MAC Ethernet com o computador hospedeiro; (*ii*) unidade de controle; (*iii*) controlador de memória DDR2 [REI09]. O computador hospedeiro faz o envio dos códigos das aplicações para a memória DDR2, a qual atua como um repositório de tarefas. O computador hospedeiro pode enviar comandos para que o MPSoC inicie a execução das aplicações ou requisitar informações de depuração. A unidade de controle é responsável por gerenciar o acesso à memória externa ou ao MPSoC.

A precisão de ciclo de relógio e os dados reportados alcançados com este modelo penaliza o tempo de simulação da plataforma, dificultando a depuração do software e disponibiliza poucas ferramentas para a análise do comportamento do sistema, como formas de onda e asserções. Visando manter a precisão temporal, mas facilitar a depuração do sistema, foi implementada uma versão da plataforma HeMPS em RTL-SystemC.

B. Modelagem em SystemC

A modelagem RTL-SystemC contém a mesma estrutura do modelo RTL-VHDL. A NoC, DMNI (*Dynamic Memory access and Network Interface*) e os módulos de memória foram escritos em SystemC-RTL. O processador é descrito utilizando um ISS (*Instruction Set Simulator*) com precisão de ciclo de relógio, conectado a um módulo SystemC (*wrapper*).

Na presente Tese a precisão de ciclo de relógio foi verificada de duas formas. Inicialmente, foi analisada e comparada, via forma de ondas, a injeção de tráfego de dados na rede. Em seguida, foi comparado o tempo de execução de diferentes tarefas. Ambos os métodos de verificação demonstraram a equivalência de comportamento entre os modelos RTL-VHDL e RTL-SystemC. Notou-se uma pequena diferença no tempo de execução (<1%), devido a dependência de dados, o que causa bolhas nos estágios do *pipeline*, e devido às instruções de multiplicação e divisão (no modelo SystemC o tempo de execução é fixo, enquanto que no modelo VHDL o tempo de execução destas instruções depende de quando os dados serão consumidos).

O modelo RTL-SystemC permite uma simulação precisa para sistemas maiores, em um tempo aceitável de simulação [PET12]. Além de formas de onda e asserções (se utilizado um simulador SystemC com interface gráfica, como ModelSim), é possível obter ganhos na depuração do sistema, através da instrumentalização do código (por exemplo, inserção de códigos de teste no ISS).

C. Modelo com Alto Nível de Abstração

O OVP é uma extensão da linguagem C que possibilita a modelagem de plataformas em alto nível de abstração. OVPsim é um simulador que traduz instruções binárias dinamicamente (DBT –

Dynamic Binary Translation) em tempo-real (JIT – *Just-in-Time*), podendo atingir uma velocidade de simulação de até 100 MIPS. O simulador OVPsim reporta o número de instruções executadas por processador, não havendo precisão temporal como nos modelos anteriores.

A Figura 8 apresenta a arquitetura da plataforma OVP, com a interconexão do processador com a NoC, ambos modelados em OVP. A NoC OVP é implementada em C (através das APIs *ppm* e *bhm*), com o mesmo algoritmo de roteamento e política de arbitragem das versões de mais baixo nível de abstração. Para o envio e recebimento de pacotes são reservados dois espaços de memória dedicados para a NoC: *reg_bank* e *buffer_proc*. O *reg_bank* é responsável pelo armazenamento dos pacotes de saída, enquanto o *buffer_proc* é responsável pelo recebimento dos pacotes de entrada. Para definir a área de endereçamento correspondente a NoC, em todo acesso à leitura e escrita nesses espaços dedicados de memórias, são executadas funções denominadas de *callbacks*.

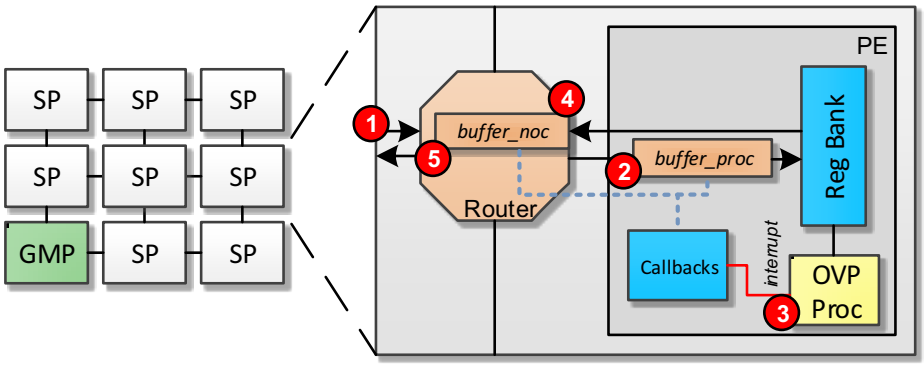


Figura 8 - Arquitetura da plataforma OVP. Fonte: [MAD15]

No recebimento de um pacote (número 1 da Figura 8), por um determinado roteador, os dados do pacote são armazenados no *buffer_proc* (número 2). Uma função *callback* interrompe o processador (número 3) assim que o recebimento do pacote for completamente efetuado. A interrupção é disparada e uma ISR (*Interrupt Service Routine*) é executada para efetuar a leitura do pacote. Quando o processador escrever algum dado no banco de registradores (Reg Bank), uma segunda função *callback* é executada pelo roteador, a qual é responsável pelo armazenamento do pacote no *buffer_noc* (número 4). Assim que o pacote estiver totalmente armazenado no *buffer_noc*, o roteador injeta-o na NoC (número 5).

Como citado anteriormente e apresentado na Figura 7, o modelo OVP disponibiliza uma maior velocidade de simulação sem fornecer informações de desempenho (como tempo de execução, energia e área), sendo também um modelo sem precisão de ciclo de relógio. Para possibilitar a estimativa de consumo de energia, mesmo em modelagens de alto nível de abstração, é capturado o volume de dados em cada canal de comunicação dos roteadores da NoC. Assim, ao término da simulação, o OVP informa o número total de instruções executadas por processador e a NoC informa o volume de comunicação em cada roteador. Dessa forma, pode-se estimar o consumo de energia e o tempo de execução por processador.

3.2.1.2 Modelo de Software

As plataformas apresentadas possuem diferentes sistemas operacionais (*µkernel*) que são executados nos GMPs, LMPs e SPs. Os GMPs e LMPs (processadores gerentes) não executam tarefas de usuários (aplicações). Os *µkernels* (sistemas operacionais) dos GMPs e LMPs são responsáveis pela gerência das tarefas, como monitoramento da NoC, mapeamento de tarefas, migração de tarefas e reclusterização. O GMP é responsável por acessar dispositivos externos (repositório de tarefas) e selecionar qual *cluster* irá executar determinada aplicação. O *µkernel* dos SPs tem duas funções primárias: escalonamento, responsável pela execução de múltiplas tarefas alocadas em sua memória local; e comunicação entre tarefas. A comunicação entre as tarefas ocorre por troca de mensagens (MPI embarcado), com duas funções principais: *send()* e *receive()*. A função *send()* (não-bloqueante) insere mensagens nas filas de comunicação, enquanto a função *receive()* (bloqueante) lê os dados das filas de comunicação.

Todas as aplicações do MPSoC HeMPS-ML (*HeMPS Multi-Level*) são modeladas como um grafo de tarefas, onde os vértices representam tarefas e as arestas representam a comunicações entre as tarefas da aplicação. As tarefas sem dependências de recepção de dados são denominadas *iniciais* (tarefa A na Figura 9). A Figura 9 exemplifica uma aplicação modelada de acordo com esta abordagem.

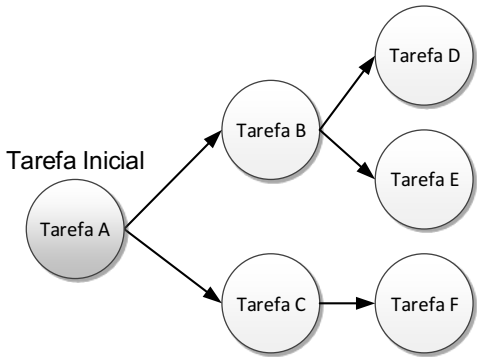


Figura 9 - Exemplo de aplicação modelada por um grafo de tarefas.

A inicialização dos recursos do MPSoC é de responsabilidade do MP. No momento em que inicia a simulação, o GMP envia pacotes de inicialização de *cluster* para os LMPs que gerenciam os mesmos. Neste pacote está inserida a informação de qual região que cada LMP gerenciará. Após o recebimento do pacote, o LMP envia pacotes de inicialização a todos os SPs da região de gerência. O mapeamento das tarefas da aplicação é realizado em tempo de execução. Dessa forma, quando uma nova aplicação deve ser inserida no sistema para execução, o GMP seleciona o *cluster* que receberá a aplicação. Após, o GMP efetua leituras no repositório de tarefas para buscar a descrição da aplicação, e transmite-a para o LMP que gerencia o *cluster* selecionado. Assim que o LMP receber o pacote com toda a descrição da aplicação, inicia-se o processo de mapeamento de tarefas. As primeiras tarefas a serem mapeadas são as tarefas iniciais (sem dependência de outras tarefas para iniciar sua computação). Quando uma tarefa inicial efetuar uma comunicação com uma tarefa ainda não mapeada, um serviço de alocação de tarefas é transmitido para o LMP, que

executará a heurística de mapeamento para esta tarefa. Todos os SPs podem executar um número parametrizável de tarefas simultaneamente.

Para que todos os *µkernels* (GMPs, LMPs e SPs) funcionem de forma unificada com todos os modelos de hardware, desenvolveu-se uma HAL. HAL é uma camada responsável pelo gerenciamento de quais instruções do *µkernel* serão executados por cada modelo de *hardware* (VHDL, SystemC ou OVP). De acordo com Yoo and Jerraya [YOO03], HAL é uma camada aplicada ao *software* capaz de torná-lo independentemente do nível de *hardware*. A Figura 10 apresenta uma visão geral da plataforma, posicionando a camada HAL entre as modelagens de *hardware* e *software*.

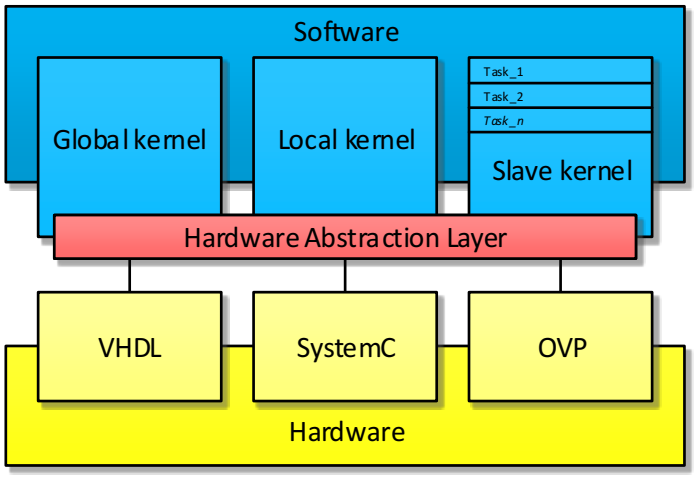


Figura 10 - Visão unificada do software da HeMPLS-ML.

A HAL possibilita a execução de um sistema operacional e de aplicações com qualquer nível de abstração de *hardware*. Dessa forma, a adoção de uma HAL simplifica a adaptação de sistemas operacionais com novos modelos de hardware.

A Figura 11 apresenta dois trechos de código da camada HAL. O primeiro exemplo (Figura 11(a)) configura o endereço inicial de uma determinada tarefa na memória (*offset*). No modelo OVP, o endereço inicial é reservado para o *buffer_proc* e para o banco de registradores. O segundo exemplo (Figura 11(b)) é a forma como o MP_G acessa o repositório de tarefas. Na implementação OVP o repositório de tarefas é tratado como um vetor (*repository[]*), enquanto nos modelos RTL (VHDL e SystemC) o repositório é uma memória externa, mapeada após o endereço 0x10000000.

a)

```
#ifdef OVP
   tcbs[i].offset = 0x20000000 + PAGESIZE*(i + KERNELPAGECOUNT);
#else
   tcbs[i].offset = PAGESIZE*(i + KERNELPAGECOUNT);
#endif
```

b)

```
#ifdef OVP
       repotask = (TaskRepository*)(&repository[appstype[appID]*APP_REPO_SIZE]);
       taskID = (appID << 8) | repotask->id;
#else
       repotask = (TaskRepository*)(0x10000000 + (appID*APP_REPO_SIZE*4));
       taskID = repotask->id;
#endif
```

Figura 11 - Exemplo de trechos de código da camada HAL.

A geração das plataformas da HeMPS-ML é feita através de um processo automatizado, desenvolvido em Python e Perl. A Figura 12 mostra um exemplo de um arquivo de geração de plataformas, ao qual habilita a criação do cenário a ser simulado. O arquivo permite ao projetista configurar diversos casos de teste para validar e avaliar diferentes parâmetros de MPSoC e diferentes cargas de trabalho (aplicações). O arquivo contém um conjunto de parâmetros: (1) nome do projeto, nome da pasta que irá conter todos os dados simulados; (2) modelo de *hardware* a ser utilizado (VHDL/SystemC/OVP); (3) tamanho do MPSoC; (4) tamanho dos *clusters*; (5) quantidade de páginas por SP; (6) endereço do processador que irá gerenciar o sistema (GMP); (7) conjunto de aplicações a serem avaliadas.

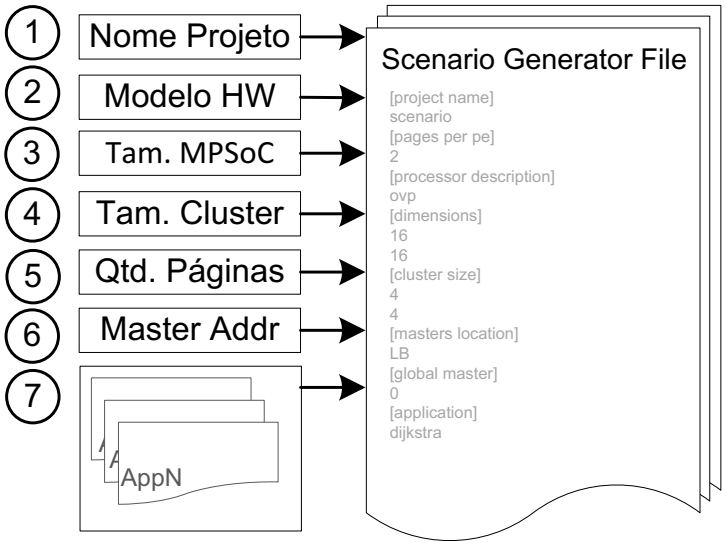


Figura 12 - Exemplo de um arquivo de geração automatizada de plataformas.

3.2.2 Plataformas Virtuais com Diferentes Organizações de Memória, sem uKernel

A interação entre os componentes do MPSoC é dada pela API de comunicação, a qual programa a interface de comunicação em um nível abstrato. Em arquiteturas de memória compartilhada, a API POSIX PThread implementa o paralelismo das aplicações usando *threads* ou processos, compartilhando dados entre si. Em arquiteturas de memórias distribuídas, a API MPI programa o paralelismo com trocas de mensagens. Neste contexto, esta Seção apresenta duas plataformas com diferentes organizações de memória, sem sistema operacional para gerência dos recursos: (*i*) MPSoC com arquitetura de memória compartilhada; (*ii*) MPSoC com arquitetura de memórias distribuídas. A segunda plataforma, com a mesma arquitetura da HeMPS-ML, foi desenvolvida para efetuar um comparativo justo com o MPSoCBench-NoC, o qual não possui sistema operacional.

3.2.2.1 Plataforma com Arquitetura de Memória Compartilhada

Esta Subseção apresenta um *framework* de geração de MPSoCs com arquitetura de memória compartilhada, denominado MPSoCBench [DUE14]. Os periféricos e processadores do

MPsoCBench são modelados em alto nível de abstração utilizando a ADL ArchC, no qual o projetista parametriza diversas funcionalidades arquiteturais do MPSoC, tais como: (*i*) tipo de processador; (*ii*) arquitetura de intercomunicação; e (*iii*) quantidade de processadores. A comunicação entre os processadores ocorre através de uma memória compartilhada, utilizando a API *acPThread* que implementa função de gerência de *threads*, sem suporte a troca de mensagens.

A *contribuição* da presente Tese foi implementar a NoC Hermes [MOR04] como nova interface de comunicação, parametrizável, no MPSoCBench [DUE14].

As aplicações do MPSoCBench são programadas a partir do modelo de programação POSIX PThread embarcado, denominado acPThread. O acPThread é uma API para sistemas embarcados contendo um conjunto limitado de funções. A API implementa funções de gerência de *threads* (`pthread_create`) e acessos à memória compartilhada (`pthread_barrier_init`). Também, implementa uma fila de controle de *threads* (*enqueue* e *dequeue*) com suporte para gerência de espaço de memória para cada *thread* (`pthread_malloc`).

A Figura 13 apresenta uma visão simplificada de um MPSoC com dimensão 6x6 e arquitetura de memória compartilhada. A NoC é modelada em SystemC-TLM2.0. A interface de comunicação entre a NoC e os processadores (modelados em ArchC) é realizada através de *wrappers*. Cada PE conectado ao roteador da NoC é composto por um *wrapper*, um processador ARM modelado em ArchC e uma cache de instruções. Um dos PEs é responsável pela inicialização das aplicações (*Manager PE* - PE_M). A plataforma contém uma memória compartilhada de 512MB, conectada através de um canal TLM ao roteador. Um terceiro PE conecta a NoC ao *locker*. O *locker* é responsável pela implementação de semáforos, *mutexes*, barreiras e variáveis condicionais que controlam o acesso à memória, garantindo a atomicidade dos dados do sistema.

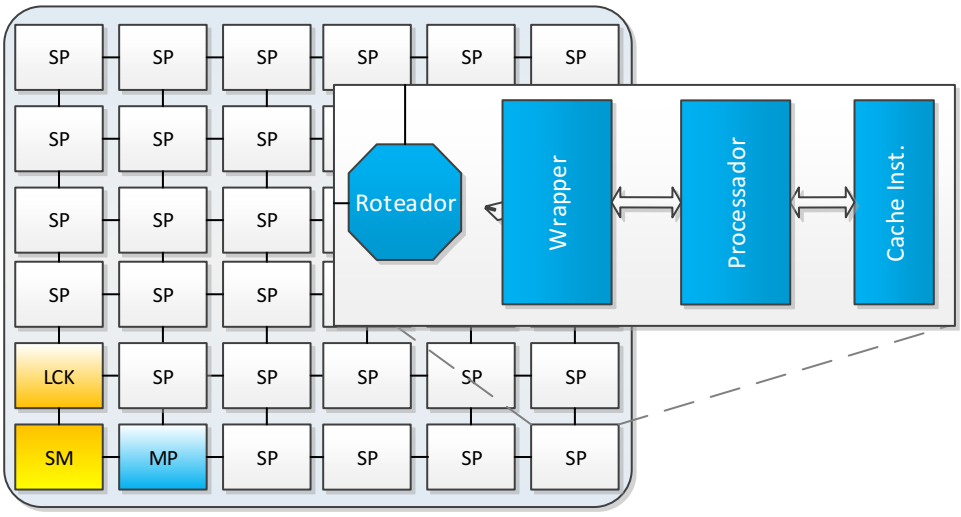


Figura 13 - Plataforma com arquitetura de memória compartilhada, implementada com a ADL ArchC.

A Figura 14 detalha a sequência de execução dos eventos de uma determinada aplicação

(app_{exec}) no MPSoCBench, com uso de NoC. Todos os PEs, exceto o PE_M, iniciam no estado *waiting*. Inicialmente (1 na Figura), a memória compartilhada recebe a descrição da app_{exec}. Depois, o PE_M recebe um conjunto de instruções da aplicação, iniciando assim sua execução. Quando o PE_M executa um *create_thread* (2), a *acPThread* armazena a ID do *thread* em uma fila de *threads* na memória compartilhada (3). Depois de efetuar a atualização da fila de *threads*, a API configura uma variável de controle que habilita a execução do *thread* por um determinado PE (4). O PE_M envia um pacote para o PE que executará a tarefa (5). Quando o PE habilitado a executar a *thread* recebe o pacote, ele deixa o estado *waiting* (6) e requisita um conjunto de instruções do *thread* à memória compartilhada (7). As instruções do *thread* são armazenadas na *cache* de instruções, e o *thread* inicia sua execução (8). A inclusão da *cache* de instruções reduz o tráfego de pacotes na NoC.

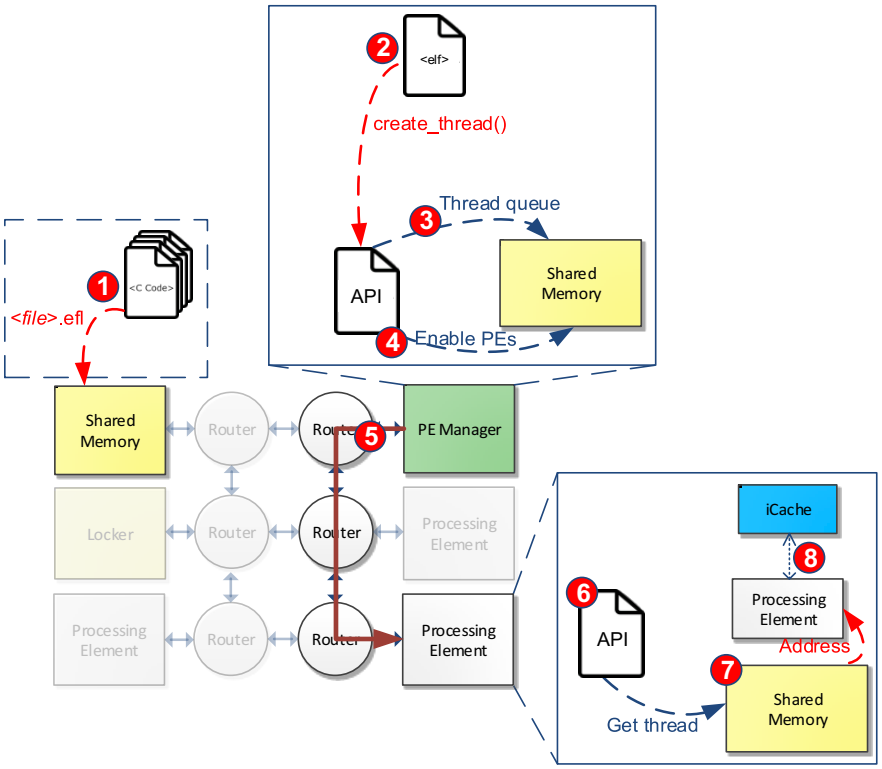


Figura 14 - Sequência de execução do MPSoCBench-NoC. Fonte: [MAD16A]

3.2.2.2 Plataforma com Arquitetura de Memórias Distribuídas

A plataforma com arquitetura de memórias distribuídas a ser avaliada e comparada com a plataforma com arquitetura de memória compartilhada, contém características arquiteturais idênticas à plataforma HeMPS-ML, apresentada na Seção 3.2.1. A diferença está na gerência de recursos, pois nesta plataforma não há sistema operacional.

A Figura 15 apresenta a sequência de geração e avaliação do MPSoC com arquitetura de memórias distribuídas sem sistema operacional. A primeira etapa é a geração da plataforma com diferentes aplicações e diferentes configurações arquiteturais, tal como a HeMPS-ML, compilando

o *software* (códigos das aplicações) e o *hardware* (processadores e periféricos modelados em OVP). Na segunda etapa, as descrições das aplicações são armazenadas na memória. Nesta etapa é importante salientar que, ao contrário da HeMPS-ML, o mapeamento é estático, onde cada PE recebe apenas uma tarefa para executar. A terceira etapa inicializa todos os roteadores, habilitando a comunicação entre todos os processadores. Na quarta etapa, o OVP reporta ao final da simulação detalhes relacionados ao número de instruções executados por todos os processadores, e o volume de comunicação na NoC.

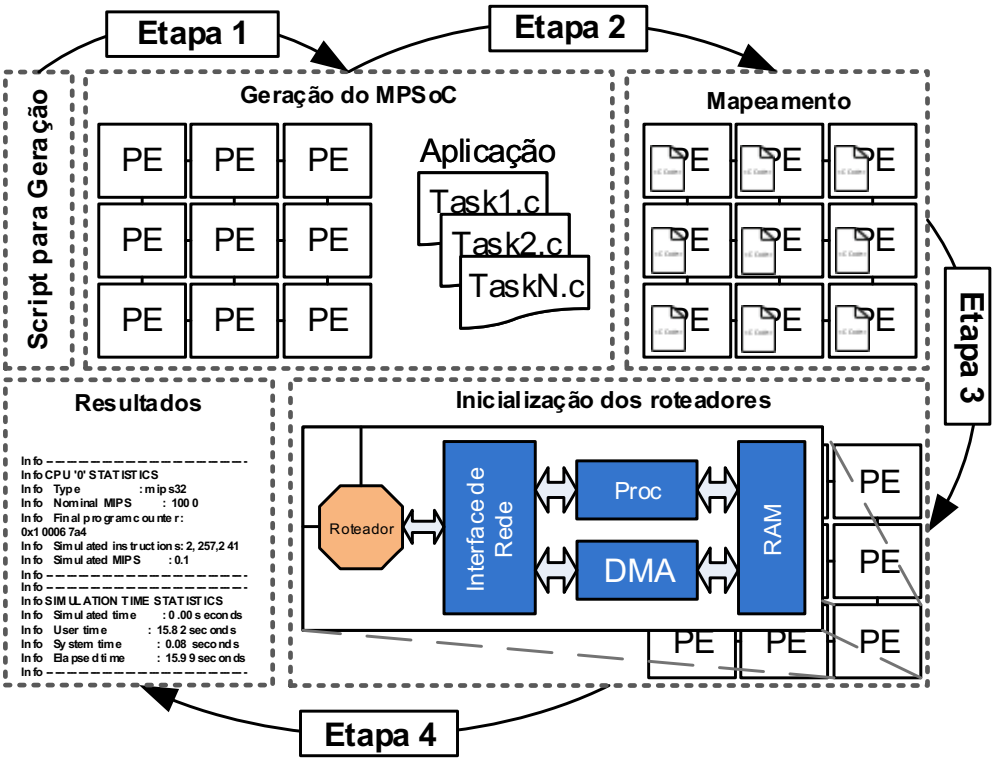


Figura 15 – Geração e avaliação do MPSoC HeMPS sem sistema operacional.

4 AVALIAÇÃO DAS PLATAFORMAS

O presente Capítulo apresenta a validação das plataformas descritas no Capítulo 3. Primeiramente será avaliada a HeMPS-ML, analisando o comportamento do uso de sistemas em diferentes níveis de abstração. Desta maneira, verificaremos o correto funcionamento do modelos, utilizando OVP. Após, são avaliados os modelos com diferentes hierarquias de memória. Assim podemos verificar, através do uso de modelos em alto nível de abstração (OVP e ArchC), os limites dos sistemas descritos e, desta forma, tomarmos as decisões de projeto para a modelagem da plataforma de referência da presente Tese.

4.1 Aplicações Utilizadas na Validação das Plataformas

Para a validação das plataformas desenvolvidas foi necessário o desenvolvimento de benchmarks para sistemas embarcados, com utilização de diferentes organizações de memória. Como já apresentado, benchmarks para arquiteturas de memória compartilhada já existem e foram adaptados na plataforma MPSoCBench, com utilização da NoC. Desta forma, para validar as plataformas com as mesmas aplicações, foi necessário o desenvolvimento dos mesmos benchmarks em MPI embarcado, sendo apresentado da Seção 4.1.1. Outros benchmarks utilizados para validação da HeMPS-ML são apresentados na Seção 4.1.2.

4.1.1 Benchmark Suite (MP2I-BencH)

Como mencionado no Capítulo anterior, o MPSoCBench é composto por aplicações adaptadas e descritas por [WOO95] (SPLASH-2), [GUT01] (MiBench) e [IQB10] (ParMiBench). Sendo um MPSoC baseado em arquitetura de memória compartilhada, os *benchmarks* do MPSoCBench são programados com a API POSIX PThread. Para que as mesmas aplicações executem em ambientes com arquitetura de memórias distribuídas foi necessário a adaptação dessas aplicações para executarem com troca de mensagens (MPI).

Com o objetivo de efetuar um comparativo entre as plataformas com diferentes hierarquias de memória, algumas aplicações desenvolvidas em PThread foram migradas para MPI. Todas as aplicações adaptadas nesta Tese podem ser geradas automaticamente com graus de granularidades diferentes. Assim, pode-se separar as aplicações com alto nível de comunicação ou computação, dependendo da avaliação necessária. Cinco aplicações dos benchmarks citados anteriormente foram adaptadas para executarem na HeMPS-ML. A Tabela 4 apresenta as aplicações desenvolvidas.

As aplicações são modeladas como aplicações mestre/escravo. Dessa forma, o mestre executa a computação inicial para a sequência da aplicação. Após efetuar as computações iniciais, o mestre paraleliza a aplicação dividindo as demais computações entre as tarefas escravas. Essa divisão pode ser feita utilizando a API PThread ou a API MPI. Se a aplicação for baseada em *thread*,

a tarefa mestre divide a computação de uma função em *n* escravos. Caso a aplicação for baseada em MPI, a aplicação é dividida em *n* tarefas comunicantes. A Figura 16 apresenta a sequência de execução deste tipo de aplicação, mestre/escravo. Inicialmente, na etapa 1 o mestre inicializa as variáveis e executa a computação inicial. Na etapa 2, o mestre divide os trabalhos a serem computados e envia os dados necessários às tarefas escravas, que iniciam sua computação na etapa 3. Ao término da execução do algoritmo, as tarefas escravas retornam o resultado calculado ao mestre (etapa 4), que na etapa 5 imprime os resultados na tela na medida em que os mesmos são recebidos.

Tabela 4 - Lista de aplicações e suas características.

Aplicações	Benchmark	API		Características
		PThread (MPSoCBench)	MPI (HeMPS-ML)	
Dijkstra	ParMiBench	✓	✓	Calcula o menor caminho entre todos os pares de um grafo representado por uma matriz adjacente. O paralelismo da aplicação ocorre por uma estratégia de decomposição de dados em que um processador calcula um limite de combinações dos pares do grafo.
FFT	SPLASH-2	✓	✓	*Fast Fourier Transform* é um algoritmo para computar o DFT (*Discrete Fourier Transform*) e seu inverso. O paralelismo ocorre via três matrizes transpostas com comunicação entre todos-para-todos os processadores do sistema.
Susan	ParMiBench	✓	✓	Algoritmo de reconhecimento de imagens de ressonância magnética do cérebro. Essa aplicação contém três diferentes formas de reconhecimento das imagens, todas configurações parametrizadas. O paralelismo ocorre por decomposição de dados, distribuídos por um nodo mestre aos demais processadores.
StringSearch	ParMiBench	✓	✓	Procura um padrão (*string*) em um determinado número de frases podendo aplicar comparações sensitivas ou insensitivas. O paralelismo ocorre pela divisão de padrões a serem encontrados.
BasicMath	ParMiBench	✓	✓	Aplicação para cálculo matemático como resolução de função cúbica, conversão de ângulo em graus para radianos e raiz quadrada de números inteiros. O paralelismo da aplicação ocorra por partição de dados.

4.1.2 Benchmark para a plataforma HeMPS-ML

Esta Seção apresenta as aplicações utilizadas para teste e validação da plataforma HeMPS-ML. Todas as aplicações são modeladas utilizando uma API de MPI embarcado, para troca de mensagens. As aplicações são modeladas na linguagem de programação C.

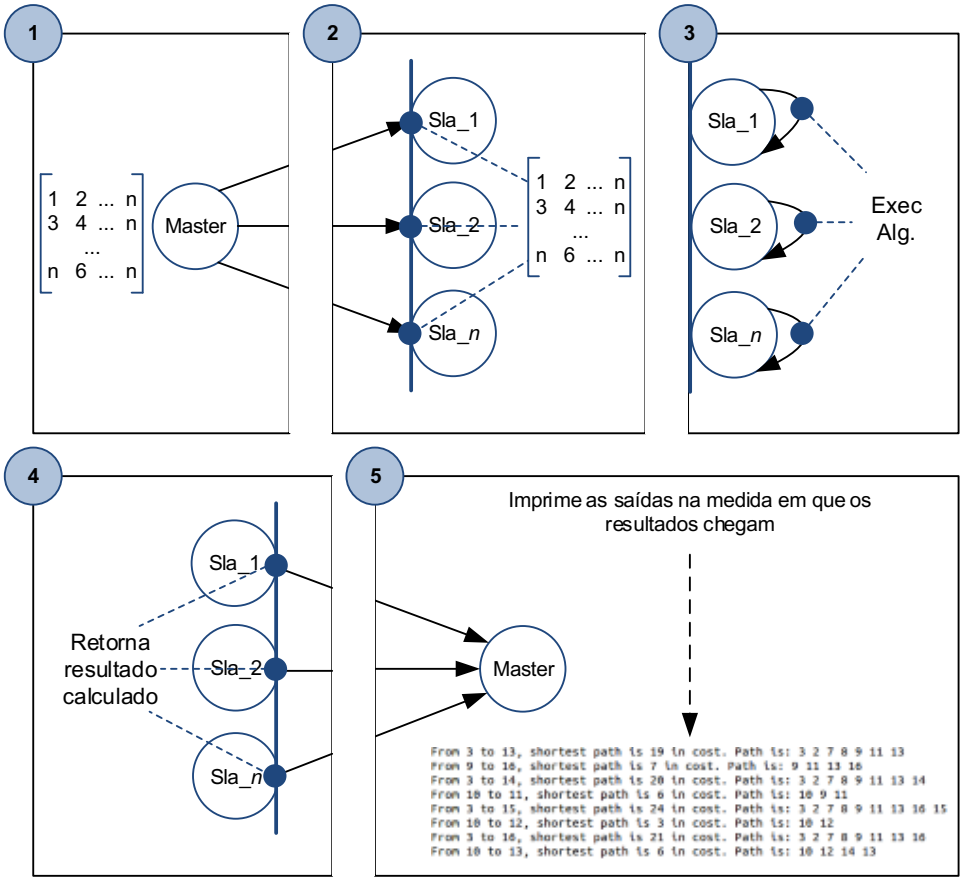


Figura 16 - Fluxo de execução de aplicações mestre/escravo.

Apresenta-se a seguir as aplicações, apresentando suas características e seus grafos de tarefas:

- **DTW** (*Dynamic Time Warping*). Dado um conjunto de padrões de referência (*templates*) a aplicação deve encontrar qual deles se ajusta melhor a um padrão desconhecido. A aplicação é divida em seis tarefas comunicantes, conforme apresentado na Figura 17(a). A tarefa "bank" é a tarefa inicial da aplicação e é responsável pela distribuição da carga de trabalho a ser computada pelas tarefas de processamento ("P1", "P2", "P3" e "P4"). Por fim, a tarefa "Rec" (*recognizer*) recebe os resultados da computação e imprime-os.
- **MPEG** (*Moving Picture Expert Group*). Aplicação que decodifica padrões de transmissão e compressão de informações de vídeo. A aplicação é formada por cinco tarefas comunicantes e pode ser analisada conforme a Figura 17(b). A aplicação inicia com a tarefa "Start" e, de forma sequencial, finaliza sua computação através da tarefa "Print".
- **VOPD** (*Video Object Plane* Decoder). Aplicação que simula a iteração entre os módulos de *hardware* de um codificador VOP (*Video Object Plane*). Esta aplicação é formada por doze tarefas, como apresentado na Figura 17(c). O início de sua execução é efetuado de forma paralela com as tarefas VLD e ARM. Esta é uma aplicação sintética, onde a computação e a comunicação são emulados por tempo de processamento e o volume de dados transmitidos, respectivamente.

- **Fixed-Based**. Aplicação de autenticação de imagens por análise espectral. A aplicação é formada por quatorze tarefas, conforme apresentado na Figura 17(d). As tarefas P1 e P2 são as tarefas iniciais.

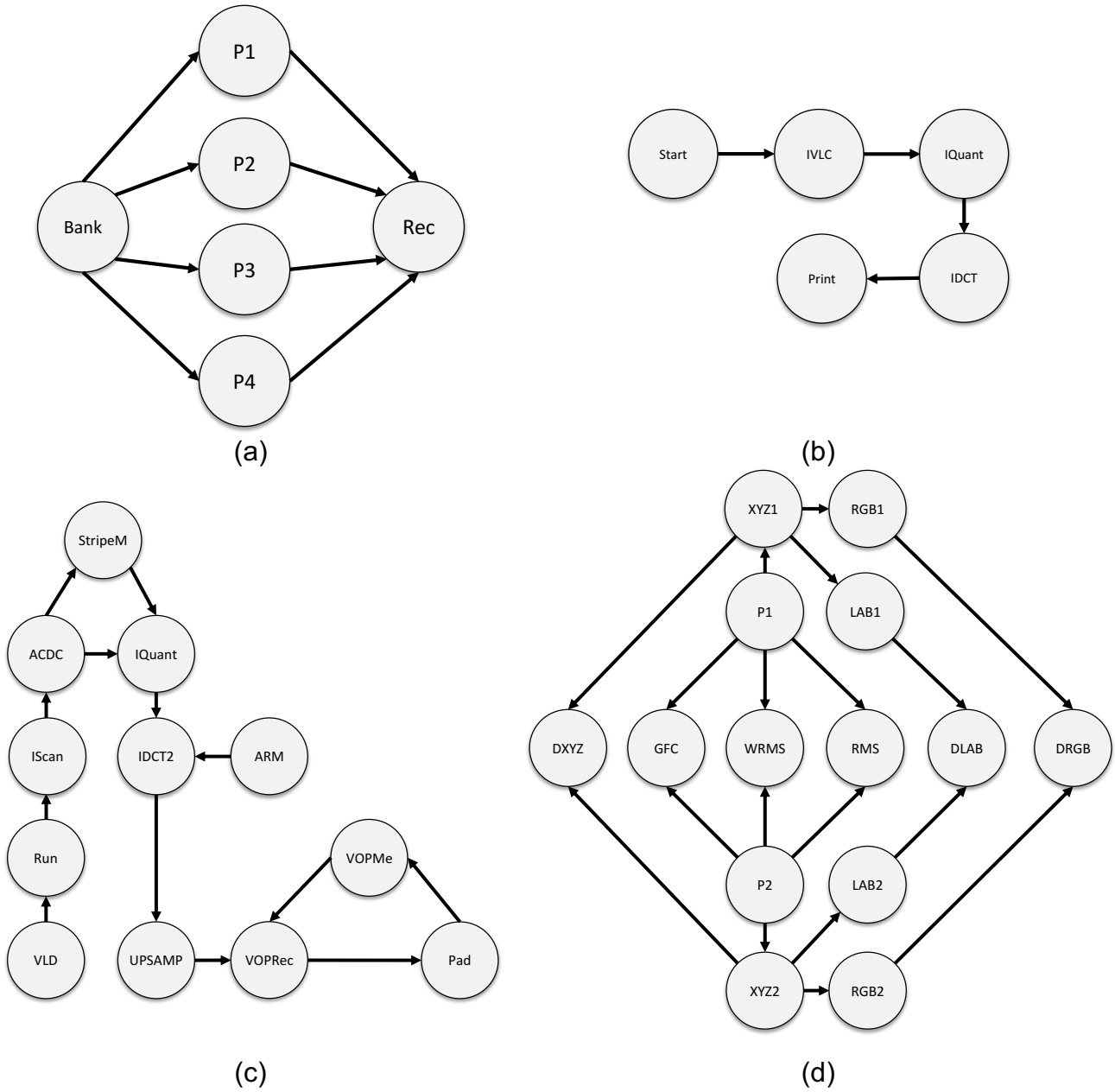


Figura 17 - Grafo sde tarefas das aplicações: (a) DTW, (b) MPEG, (c) VOPD, Fixed-Based.

4.2 Avaliação da HeMPS-ML

Esta Seção apresenta comparativos entre as três plataformas modeladas. O comparativo é dividido em três partes: *A*- comparativo entre os modelos RTL (VHDL e SystemC); *B*- comparativo entre os modelos SystemC e OVP; *C*- comparativo com os três modelos. Como mencionado, o modelo SystemC apresenta características de modelagem RTL, como precisão de ciclo de relógio. Também, este modelo apresenta características de modelagens em alto nível de abstração, uma

melhor depuração do *software* e menor tempo de simulação, comparado ao modelo VHDL.

A. Comparativo entre os modelos RTL – VHDL e SystemC

Esta Subseção compara o tempo de simulação e o número de instruções executadas para os dois modelos RTL.

A Tabela 5 apresenta a configuração de três cenários utilizados para avaliação dos modelos RTL. O MPSoC modelado é de tamanho 4x4 com um PE responsável pela gerência do sistema (GMP) e quinze PEs responsáveis pela execução das tarefas (SPs). Foram utilizadas duas aplicações: MPEG e VOPD. O primeiro cenário avaliado (C1) executa duas aplicações MPEG e uma aplicação VOPD, totalizando 22 tarefas. Para avaliar o comportamento das plataformas com as mesmas características, no cenário dois (C2) e três (C3) aumentou-se a quantidade de tarefas executas para 44 e 88, respectivamente.

Tabela 5 - Configuração dos cenários para avaliação dos modelos RTL.

Aplicações	C1	C2	C3
MPEG (5 tarefas)	2	4	8
VOPD (12 tarefas)	1	2	4
Total de tarefas	**22**	**44**	**88**

A Tabela 6 apresenta a avaliação do tempo de simulação para os três cenários. A utilização do modelo SystemC resultou em um *speedup* de duas ordens de grandeza comparado ao tempo de simulação do modelo VHDL.

Tabela 6 - Tempo de simulação (em segundos) dos modelos RTL e SystemC.

Cenários	VHDL	SystemC	Speedup VHDL/SystemC
C1	2.425	19	**127**
C2	4.407	31	**142**
C3	7.932	51	**155**

A Tabela 7 apresenta a avaliação do tempo de execução necessário para executar todas as tarefas para cada cenário. A Tabela 8 apresenta o número total de instruções executadas por todos os processadores das plataformas. Os resultados analisados demonstram que o modelo SystemC e o modelo RTL apresentam comportamento similar. Observou-se uma pequena diferença no tempo de execução (<1%) devido a simplificações no ISS, não refletindo a real operação do processador modelado em VHDL.

Tabela 7 - Tempo de execução das aplicações (em ciclos de relógio) dos modelos VHDL e SystemC.

Cenários	VHDL	SystemC	VHDL/SystemC
C1	265.015	265.228	**0,998**
C2	526.660	528.524	**0,996**
C3	1.050.247	1.055.543	**0,995**

Tabela 8 - Número total de instruções executadas nos modelos VHDL e SystemC.

Cenários	VHDL	SystemC	VHDL/SystemC
C1	422.757	423.120	**0,999**
C2	834.335	836.335	**0,997**
C3	1.662.311	1.664.941	**0,998**

B. Comparativo entre os modelos SystemC e OVP

A Subseção anterior demonstrou a precisão do modelo SystemC em termos de instruções executadas e tempo de execução, usando o modelo VHDL como referência. Nesta Subseção o modelo SystemC é o modelo de referência para as avaliações de comportamento da plataforma OVP para MPSoCs de maior dimensão. Considerando que o modelo VHDL apresentou resultados similares em relação ao modelo SystemC, nesta Subseção o modelo VHDL não é utilizado, dado o elevado tempo para simular MPSoCs de maior dimensão.

A Tabela 9 apresenta a configuração de cinco cenários utilizados para avaliar os modelos SystemC e OVP. As duas primeiras linhas da tabela correspondem ao tamanho do MPSoC e dos *clusters*, respectivamente. A terceira linha apresenta o número de SPs (*Slave Processor*) e MPs (*Manager Processor*). Por exemplo, no MPSoC de tamanho 10x10 com *clusters* de tamanho 5x5, 96 PEs executam tarefas de aplicações e 4 PEs são reservados para a gerência do sistema. As linhas de 4 a 7 da Tabela 9 contém o número de aplicações executadas. No MPSoC de tamanho 10x10 são executadas duas aplicações MPEG e DTW, e três aplicações Dijkstra e Fixed-Based. A última linha apresenta o número total de tarefas executadas.

Tabela 9 – Configuração dos cenários para avaliação dos modelos SystemC e OVP.

Tamanho do MPSoC	**4x4**	**6x6**	**8x8**	**10x10**	**12x12**
Tamanho do Cluster	**4x4**	**3x3**	**4x4**	**5x5**	**4x4**
#SPs / #MP	**15/1**	**32/4**	**60/4**	**96/4**	**135/9**
$MPEG_{(5)}$	1	1	1	2	4
$Dijkstra_{(6)}$	-	1	3	3	5
$DTW_{(10)}$	1	1	2	2	4
$F.Base_{(15)}$	-	1	1	3	3
Total de Tarefas	**15**	**36**	**58**	**93**	**135**

Na modelagem da plataforma OVP, o tempo de execução é definido pelo número de instruções executadas pelo MP_G (último PE a executar instruções no MPSoC). Dado que o CPI médio do MIPS é igual a 1, o número de instruções executadas pelo GMP corresponde aproximadamente ao número de ciclos de relógio consumidos. O MP_G inicia as aplicações mapeando-as nos *clusters*. Quando uma aplicação finaliza sua execução o MP_C envia uma mensagem ao MP_G informando o término da mesma, para que seu espaço de *cluster* seja liberado para mapeamento de novas aplicações. No fim da simulação o número total de instruções executadas pelo MP_G é informado.

A Figura 18 apresenta o tempo de execução, em ciclos de relógio, para todos os cenários. Para MPSoCs com até 64 PEs, a diferença entre os modelos SystemC e OVP é aproximadamente 10%. Para sistemas maiores, esta diferença pode atingir 25%. Mesmo aplicando a mesma carga de trabalho em ambas as simulações, o número de instruções (tempo de execução) varia devido a diferenças no mapeamento, número de tarefas por processador e o tráfego de dados na NoC.

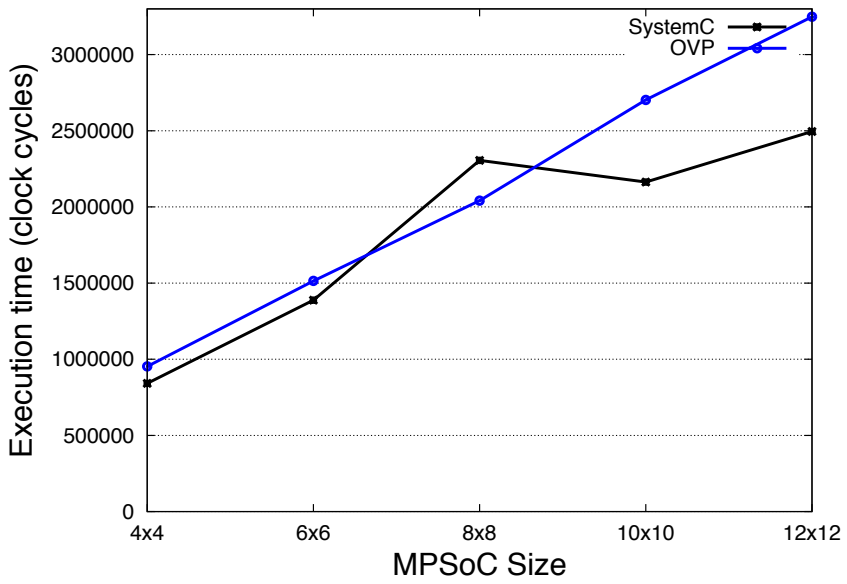


Figura 18 - Avaliação de tempo de execução dos modelos OVP e SystemC. Fonte: [MAD15]

Ao término da simulação, ambos modelos reportam o número total de instruções executadas por PE. O número de instruções executadas possibilita uma estimativa de consumo de energia, uma vez que cada instrução tem seu custo energético definido. Dessa forma, a Figura 19 apresenta o número total de instruções executadas em todos os cenários. A diferença entre MPSoCs de tamanhos maiores é de 17%.

Os resultados apresentados na Figura 19 corroboram o fato de que a simulação OVP, apesar de não ter precisão de ciclo de relógio, tem uma precisão aceitável em termos de número de instruções executadas (diferença inferior a 20%), o que permite estimativas de tempo de execução e consumo de energia em alto nível de abstração.

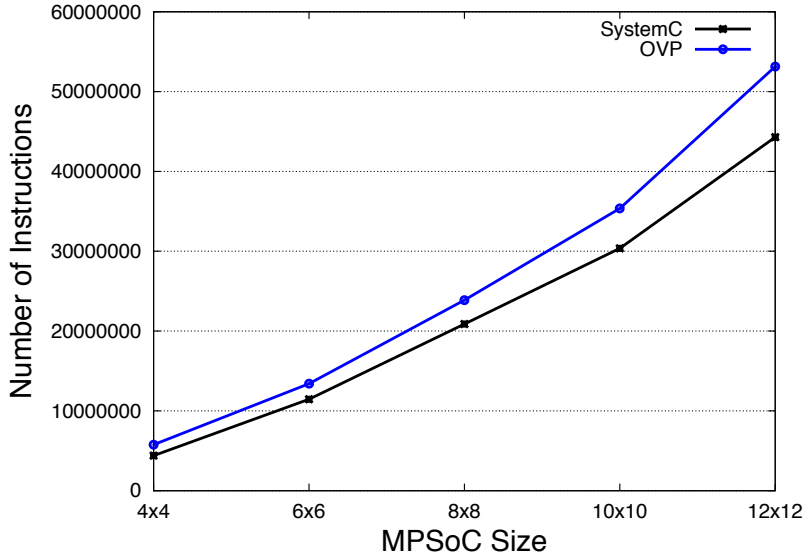


Figura 19 - Número de instruções executadas nos modelos OVP e SystemC. Fonte: [MAD15]

A Figura 20 apresenta o tempo de simulação de todos os cenários avaliados, sendo este o tempo-real requerido para finalizar a simulação. Deve-se observar que o número de tarefas para cada cenário é aproximadamente igual ao número de PEs do sistema, o que corresponde a uma carga de trabalho pequena. Mesmo com baixa carga de trabalho, a simulação OVP teve uma velocidade de simulação cinco vezes mais rápida comparada à simulação SystemC.

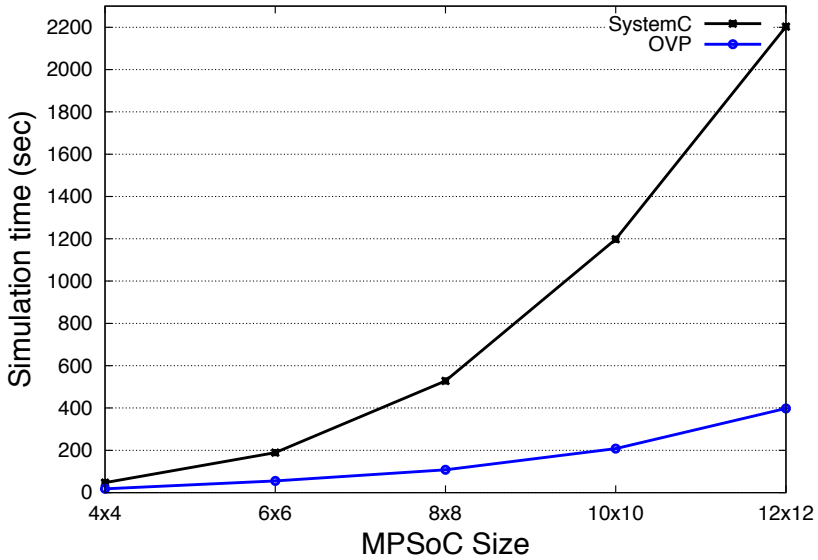


Figura 20 - Tempo de simulação dos modelos OVP e SystemC. Fonte: [MAD15]

C. Comparativo entre os três modelos de plataforma

Esta Subseção tem por objetivo apresentar um comparativo de maneira que possamos analisar as três métricas (tempo de execução, tempo de simulação e instruções executadas) para as três plataformas.

O cenário utilizado para avaliação dos modelos é configurado como segue:

- MPSoC com 16 PEs (4x4);

- Gerenciamento centralizado do sistema – 1 único MP;
- Execução de oito instâncias das aplicações MPEG e Dijkstra, cada uma composta por cinco tarefas comunicantes.

O cenário possui esta configuração por ser um MPSoC de tamanho pequeno, possibilitando a execução do mesmo em VHDL, com um tempo de simulação não muito elevado. Este cenário possibilita também a execução de aplicações com diferentes características, como: MPEG com execução sequencial de tarefas, e Dijkstra com execução paralela de tarefas.

A Figura 21 apresenta um comparativo ente os modelos avaliados. A Figura é dividida em três grupos de categorias: (*i*) fatia externa, representando o tempo de simulação; (*ii*) fatia central, representando o tempo de execução; e (*iii*) fatia interna, representando a quantidade de instruções simuladas. Os dados são divididos nas cores azul, vermelha e verde, sendo respectivamente representantes dos modelos VHDL, SystemC e OVP.

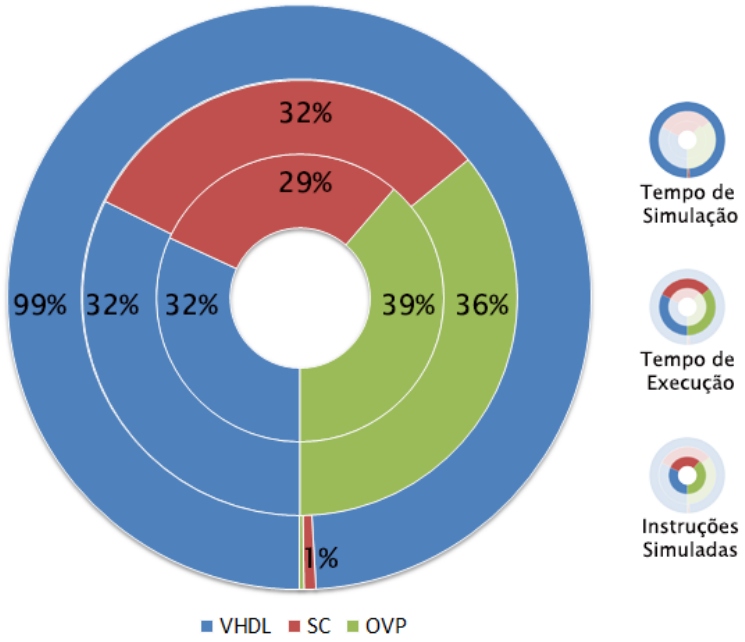


Figura 21 - Comparativo entre os modelos VHDL-SC-OVP.

Analisando o tempo de simulação dos modelos (fatia externa), podemos verificar que os modelos OVP e SystemC, juntos, representam apenas 1% do tempo total de simulação. Os 99% apresentados na fatia de dados do modelo VHDL comprovam que tal modelo apresenta um elevado tempo de simulação. Ao verificarmos o tempo de execução (fatia central), podemos analisar que o comportamento do sistema, nos três modelos de abstração, apresenta certa similaridade. A diferença apresentada pelo modelo OVP é em função do mapeamento que, em alguns momentos, pode tomar decisões diferentes dos outros modelos. O mesmo ocorre quando analisamos as instruções simuladas. Os modelos VHDL e SystemC apresentam equivalência na quantidade de instruções simuladas, enquanto o modelo OVP executa mais instruções devido a pequenas alterações comportamentais do sistema.

Através desta figura, concluímos que o modelo VHDL é indicado para validação com precisão de ciclo de relógio, e obtenção de métricas como potência dissipada e área, porém ao custo de elevado tempo de simulação. Já o modelo SystemC apresenta redução no tempo de simulação e possibilita a validação de plataformas com precisão de ciclo de relógio, tendo comportamento muito similar ao modelo VHDL. O modelo OVP é o modelo indicado para validar o software da plataforma por disponibilizar ferramentas de depuração e baixo tempo de simulação.

4.3 Avaliação das Plataformas Virtuais com Diferentes Hierarquias de Memórias

Esta Seção apresenta a avaliação da escalabilidade de plataformas com diferentes organizações de memória. Características comuns aos modelos incluem:

- PE com palavra de 32-bit e flits com 16-bit;
- Roteador: chaveamento de pacotes *wormhole*;
- Roteamento XY;
- Arbitragem round-robin centralizada (no nível do roteador, não no nível da NoC).

Ambos modelos executam duas aplicações. A primeira é a Dijkstra em três versões: (*i*) uma tarefa produtora e uma consumidora; (*ii*) uma produtora e três consumidoras; (*iii*) uma produtora e quatro consumidoras. A segunda aplicação é a FFT, que apresenta um volume de comunicação superior à Dijkstra.

Para analisar a questão de escalabilidade de MPSoCs, são efetuadas duas avaliações. A primeira avalia o volume de comunicação transferido pela NoC, mostrando a distribuição do tráfego de dados na NoC. A segunda avalia o número total de instruções simuladas pelos processadores.

Na plataforma com arquitetura de memórias distribuídas, como mencionado anteriormente, cada um dos PEs executa uma tarefa. Assim, o número de tarefas é igual ao número de PEs disponíveis no MPSoC. Os experimentos são executados com 4 (2x2), 16 (4x4), 36 (6x6), 64 (8x8), 100 (10x10), 144 (12x12), e 225 (15x15) PEs.

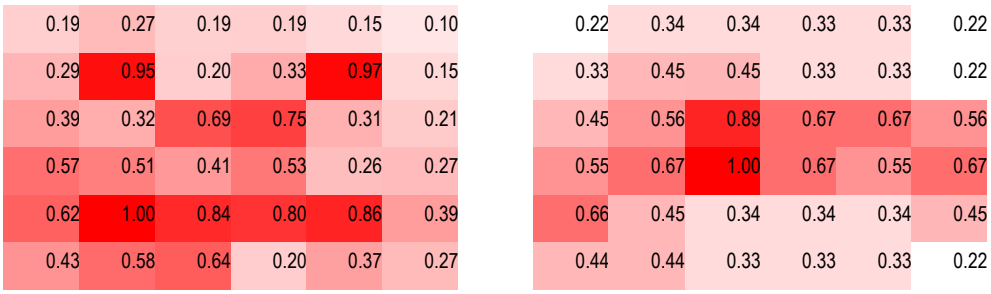
A Tabela 10 apresenta o número total de *flits* transferidos e o total de instruções executadas, para cada tamanho de MPSoC simulado. A plataforma assume que as descrições das tarefas já estão armazenadas nas memórias dos PEs. Dessa forma, todos os *flits* transferidos são *flits* de dados, sem tráfego de instruções pela NoC. Pode-se notar que o crescimento, em ambos os parâmetros analisados é praticamente linear com o tamanho do MPSoC (desvio padrão de $r^2 > 0,98$ para MPSoCs maiores que 16 PEs).

Outra avaliação importante é a distribuição do tráfego internamente ao MPSoC. A Figura 22 apresenta o volume relativo de comunicação transferido por cada roteador na plataforma de arquitetura de memórias distribuídas. Nota-se a não existência de *hotspots*, com um tráfego distribuído de forma relativamente uniforme no MPSoC (Figura 22(a)). Na Figura 22(b), o volume

relativo de comunicação é muito semelhante entre os roteadores, mostrando um tráfego de dados mais distribuído que no caso (a).

Tabela 10 - Número de flits e instruções executadas, arquitetura de memórias distribuída.

Número de PEs	Número total de *flits* transferidos (10^3 *flits*)		Número total de instruções executadas (10^6 instruções)	
	Dijkstra	FFT	Dijkstra	FFT
4	14,6	17,1	1,7	5,2
16	46,0	112,6	4,5	12,4
36	143,2	339,6	13,8	22,3
64	263,8	1084,1	49,3	40,1
100	446,7	1915,7	117,1	62,9
144	921,9	3294,6	241,0	91,1
225	1425,2	6294,5	588,1	143,0



(a) Dijkstra application (b) FFT application

Figura 22 - Volume de comunicação por roteador de um MPSoC (arquitetura de memória distribuída) 6x6. Fonte: [MAD16A]

Por outro lado, na plataforma com arquitetura de memória compartilhada o número de tarefas é diferente do tamanho do MPSoC, pois um roteador é dedicado à memória compartilhada e outro ao *locker*. Diferentemente da plataforma com memórias distribuída, devido ao longo tempo de simulação do ArchC em comparação ao OVP, são avaliados MPSoCs de tamanhos menores, mas o suficiente para apresentar os efeitos da escalabilidade.

A Tabela 11 apresenta o número total de *flits* transferidos e instruções executadas em cada MSPoC simulado. Comparado com a arquitetura de memórias distribuída, a arquitetura de memória compartilhada apresenta um comportamento diferente. Foi observado um crescimento exponencial nos parâmetros analisados. Duas razões explicam tal comportamento. Primeiro, a adoção de memória compartilhada implica em um alto número de acesso a memória e ao *locker*, para compartilhar os dados de aplicações. Com MPI este comportamento não existe, o acesso é feito em memórias locais conectadas diretamente ao processador, e a comunicação é realizada por troca de mensagens. Segundo, o uso da memória de instruções aumenta o tráfego de dados na

NoC, pois são realizados pedidos de instruções à memória compartilhada. O número de instruções executadas aumenta porque durante um *cache miss* o processador fica em laço aguardando os dados da cache. Este comportamento é característico da modelagem com ArchC. O processador poderia entrar em estado de espera (*hold*), de forma a reduzir o número de instruções executadas.

Tabela 11 - Número de flits e instruções executadas, arquitetura de memória compartilhada.

Número de PEs	**Número total de *flits* transferidos (10^6 flits)**		**Número total de instruções executadas (10^6 instruções)**	
	Dijkstra	**FFT**	**Dijkstra**	**FFT**
4	31,3	68,7	5,18	10,3
8	45,5	110,7	6,06	13,4
10	76,3	164,7	8,68	17,9
12	111,8	287,6	13,21	32,6
18	188,1	399,8	18,13	38,4
42	4075,7	9278,8	339,99	706,9

Na Figura 23 apresenta-se o volume relativo de comunicação transferido por cada roteador no MPSoC com arquitetura de memória compartilhada. Como esperado, os *hotspots* são claramente identificados próximo a memória compartilhada (SM) e ao *locker* (L). O tráfego é distribuído de forma não uniforme internamente no MPSoC. Efeitos de desgastes, como eletro-migração, pode aparecer mais rapidamente do que na plataforma de arquitetura distribuída de memória, assim reduzindo o tempo de vida do sistema.

(a) Dijkstra application (b) FFT application

Figura 23 - Volume de comunicação por roteador de um MPSoC (MC) 6x7. Fonte: [MAD16A]

Os resultados apresentados na Tabela 11 e na Figura 23 mostram as limitações da arquitetura de memória compartilhada para MPSoCs com mais de 64 PEs.

4.4 Considerações Finais

Este Capítulo apresentou as avaliações de cada plataforma modelada na presente Tese. Primeiramente apresentou-se a modelagem das plataformas, juntamente com suas APIs de programação. Para MPSoCs com arquitetura de memória compartilhada foi apresentado o

MPSoCBench, modelado com a ADL ArchC. O modelo de programação do MPSoCBench é baseado em *threads*, com utilização da API acPThread. Para MPSoCs com arquitetura de memórias distribuída foram apresentadas plataformas com gerência de recursos distribuída através de sistemas operacionais e uma segunda plataforma sem gerência de recursos. A primeira plataforma propõe uma modelagem abstrata de MPSoCs para auxiliar no desenvolvimento do *software*, com simulações rápidas e depuração eficaz. Dessa forma, apresentou-se a HeMPS-ML, modelada no em RTL (utilizando VHDL e SystemC) e no nível *untimed* (OVP). A segunda plataforma contém as mesmas características arquiteturais da HeMPS-ML, mas modelada apenas com a ADL OVP e sem sistema operacional para gerência.

Podemos concluir que a execução de plataformas com tráfego localizado (*hotspot*) é um dos fatores que podem limitar a escalabilidade de sistemas com memórias compartilhadas e induzem congestionamento da rede. Um dos maiores problemas para aplicações com restrições temporais, como aplicações de tempo-real, é o congestionamento na rede, pois este implica na não previsibilidade na comunicação. Além do congestionamento, podemos levar em consideração o tempo de vida dos componentes do sistema. A área do *hotspot* tende a ter seu tempo de vida reduzido comparado às demais áreas do sistema. Dessa maneira, a área afetada pode apresentar falhas em seus componentes diminuindo os recursos disponíveis para computação e podendo impossibilitar o acesso à memória compartilhada. Para MPSoCs de maior tamanho, o uso da arquitetura de memórias distribuídas tende a ser a alternativa adequada. Porém, não é possível executar aplicações modeladas com *thread*. Arquitetura de memória DSM (*Distributed Shared Memory*), com algumas memórias compartilhadas distribuídas entre as bordas do MPSoC, e com memórias locais distribuídas em cada PE, pode ser uma direção a seguir, podendo executar aplicações com *threads* e troca de mensagens concomitantemente.

Os modelos apresentados no Capítulo anterior, juntamente com as avaliações apresentadas neste Capítulo, expõem as vantagens e limitações de cada sistema. Também, mostram os benefícios da utilização de modelagens abstratas para auxiliar os projetistas na tomada de decisões de projetos em sistemas embarcados. Os modelos de alto nível de abstração, apesar de não serem precisos em ciclo de relógio, permitem uma estimativa rápida de dados com baixo erro (<17%) no início do projeto.

5 PLATAFORMA DE TEMPO-REAL

Este Capítulo define os conceitos de sistemas computacionais de tempo-real. Apresenta-se a parametrização de tarefas de aplicações e as funcionalidades específicas para sistemas operacionais que executam aplicações com restrições de tempo-real. Kopetz et al. [KOP97] definem sistemas de tempo-real como um sistema computacional em que a exatidão do comportamento do sistema depende não apenas dos resultados lógicos obtidos através da computação, mas também no instante físico em que os dados resultantes são produzidos.

Sistemas de tempo-real podem ser classificados como *Hard-RT* ou *Soft-RT*, dependendo das restrições definidas pela aplicação. Sistemas *Hard-RT* são sistemas que devem, obrigatoriamente, atender a todos os requisitos da aplicação. A violação destes requisitos pode causar consequências catastróficas ao ambiente sob controle. Por outro lado, sistemas *Soft-RT* são sistemas em que o atendimento aos requisitos da aplicação são desejáveis, mas a violação de alguns requisitos não causa danos ao mundo físico.

A Figura 24 apresenta exemplos de sistemas embarcados com restrições de tempo-real. Os exemplos presentes na área vermelha são de sistemas *hard-RT*, tais como:

- Em acidentes automotivos, o atraso no atendimento das restrições de tempo em um sistema de *airbags* pode ocasionar danos graves ou fatais aos indivíduos envolvidos. Já a falha de sistemas de freios ABS (*Anti-lock Breaking System*) e controladores de velocidade podem causar danos irreparáveis aos passageiros;
- Satélites controladores de tráfego aéreo, sinais de GPS (*Global Position System*) e a comunicação via satélite devem ter seus requisitos priorizados e em pleno funcionamento. Caso os requisitos de tempo não forem atendidos, problemas como acidentes aéreos e marítimos e falhas de acesso a internet podem ocorrer.
- O atraso no atendimento de requisitos de controladores embarcados em indústrias com resíduos de alta periculosidade pode causar danos ambientais que só seriam resolvidos após algumas gerações.

Os exemplos apresentados na área azul são de sistemas *soft-RT*, tais como:

- Controladores residenciais, como câmeras e alarmes, podem sofrer atrasados e violações das restrições de aplicações com requisitos de tempo não causando graves danos;
- Dispositivos eletrônicos como *tablets*, celulares e *wearable* são componentes essenciais no nosso dia-a-dia, mas quando suas funcionalidades não atendem nossa demanda, grandes problemas não ocorrerão.
- A grande maioria das compras e vendas de mercadorias e prestação de serviços é efetuada através de transações financeiras. O atraso em uma transação pode ser resolvido em uma segunda tentativa.

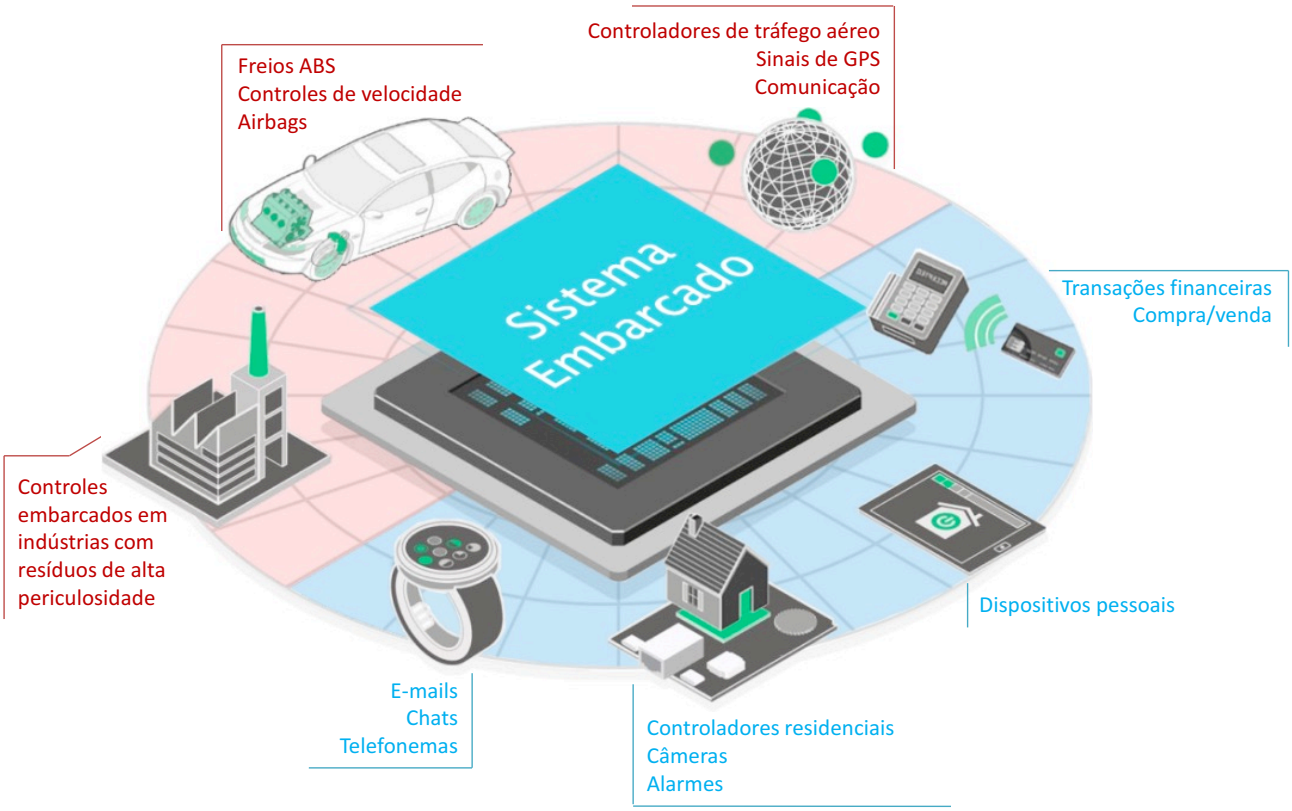


Figura 24 - Sistemas embarcados *Hard-RT* (área vermelha) e *Soft-RT* (área azul) presentes em dispositivos eletrônicos usados no nosso dia-a-dia.

Segundo Ramming et al. [RAM09] as aplicações de tempo-real são aplicações como qualquer outra, apresentando apenas uma diferença essencial: a noção de tempo. Isso significa que o sistema é dependente da computação e também do tempo que ela inicia sua execução. A característica principal de sistemas de tempo-real é que o tempo do sistema deve ser medido na mesma escala que o tempo do ambiente controlado. O principal parâmetro que caracteriza uma aplicação de tempo-real é o prazo determinado para computar algo. Este prazo é definido como *deadline*.

A Figura 25 apresenta as restrições de tempo em tarefas de aplicações tempo-real. As restrições de tarefas (t_i) são definidas por uma tupla com os seguintes parâmetros:

- C_i , representa o tempo de computação de t_i;
- T_i , corresponde ao período de t_i;
- D_i , representa o *deadline* a ser atendido por t_i;
- P_i , prioridade de t_i. Todas as tarefas de uma aplicação contêm uma prioridade. Tal característica define a prioridade de execução de tarefas em um sistema operacional com múltiplas tarefas.

Aplicações de tempo-real contêm tarefas que podem ser classificadas em dois tipos: periódicas e aperiódicas. Normalmente as instâncias computacionais das tarefas são chamadas de *jobs*. Todos os *jobs* de uma tarefa compartilham o mesmo código, portanto os períodos e os

deadlines também são compartilhados. Para o caso das tarefas periódicas os *jobs* contêm *deadlines* fixos. Quando tarefas tem *deadlines* fixos significa que todos os *jobs* desta tarefa devem finalizar sua computação no mesmo período de tempo que o primeiro *job*. Para o caso de tarefas aperiódicas o período não existe, o tempo computacional de um *job* é desconhecido e sua finalização pode acontecer em qualquer momento.

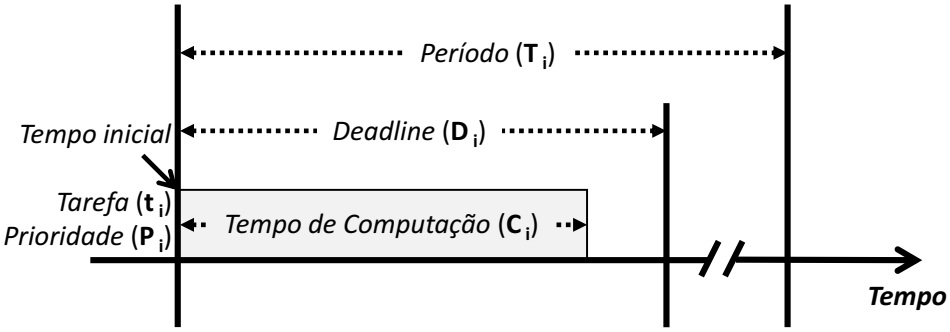


Figura 25 - Modelo de restrições de tempo para tarefas de aplicações de tempo-real. Fonte: [MAD16B]

As restrições de tempo-real das tarefas são definidas pela aplicação. Tais restrições são parâmetros essenciais para as funcionalidades do sistema operacional, tais como: escalonamento de tarefas em processadores com execução de múltiplas tarefas e o mapeamento de tarefas de aplicações com restrição de tempo-real nos recursos do sistema.

Este Capítulo apresenta a modelagem de aplicações com restrição de tempo-real na Seção 5.1. Na Seção 5.2 é apresentado o mecanismo de escalonamento preemptivo que se baseia nas prioridades das tarefas. A Seção 5.3 apresenta uma análise de escalabilidade, a qual é utilizada como função custo do mapeamento de tarefas apresentado na Seção 5.4.

5.1 Modelagem de Aplicações com Restrição de Tempo-Real

Esta Seção apresenta a modelagem das aplicações de tempo-real. Os requisitos das tarefas com restrições de tempo-real são definidos pela própria aplicação. O *µKernel* fornece suporte às aplicações para definição das restrições de tempo através de uma chamada de sistema (*syscall*) denominada `RealTimeConstraints`. Esta chamada de sistema é responsável por informar o sistema operacional sobre as restrições, caso existam, de cada tarefa das aplicações no sistema. Como apresentado no Capítulo 3, aplicações são modeladas como grafo de tarefas. A Figura 26 apresenta um grafo de tarefas exemplificando o código de uma determinada tarefa (t_A) que configura as restrições de tempo-real. Analisando o código podemos notar que na linha 2 são definidos os valores de tempo de uma tarefa periódica. Na linha 3 a chamada de sistema `RealTimeConstraints` é executada para notificar as restrições da tarefa ao sistema operacional. As linhas 4 a 8 representam o *job* da tarefa, sendo o código que será computado periodicamente conforme a configuração de iterações da aplicação. É importante salientar que a

chamada de sistema pode ser executada quantas vezes forem necessárias, modificando as restrições de tempo da tarefa em tempo de execução.

Exemplo de código de T_A

```
1  int main() {
2     int T=80100, D=4500, C=4000, P=1;
3     RealTimeConstraints(T, D, C, P);
4     for (int i=0; i<iterations; i++) {
5         msgA = [...];
6         send(msgA, taskB);
7         msgB = process(msgA);
8         send(msgB, taskC);
9     }
10    return 0;
11 }
```

Grafo de tarefas

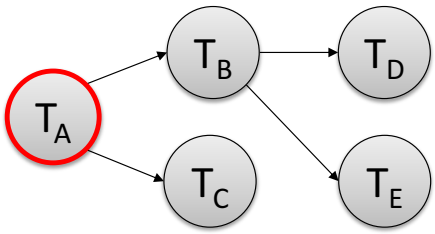


Figura 26 – Modelo de aplicação com restrições de tempo-real e exemplo de código de tarefas.
Fonte: [MAD16B]

Uma aplicação, como a modelada na Figura 26, contem *n* tarefas com diferentes requisitos de tempo. O hiper-período da aplicação é o intervalo de tempo onde são executados todos os *jobs* de uma iteração. Dessa forma, o hiper-período inclui o tempo de execução de todas as tarefas da aplicação, incluindo o tempo de comunicação na troca de mensagens entre as tarefas.

5.2 Mecanismo de Escalonamento

Esta Seção apresenta o algoritmo de escalonamento utilizado para avaliação e validação dos resultados apresentados nesta Tese. Alocar as tarefas no processador garantindo que todas, em algum momento, serão executadas é o principal objetivo dos mecanismos de escalonamento. As tarefas podem executar em tempos diferentes e em ordens diferentes, isso vai depender da função custo definida no mecanismo de escalonamento. Os algoritmos podem ser classificados em dois tipos: preemptivos e não-preemptivos. Nas abordagens preemptivas, as tarefas em execução podem ser interrompidas a qualquer momento e sua reinicialização pode acontecer em algum tempo depois. Este tipo de abordagem quando utilizado em sistemas específicos, como sistemas de tempo-real, podem gerar um atraso na execução de determinadas tarefas. Assim, o sistema deve garantir que os requisitos de tempo sejam atendidos. Para os algoritmos de escalonamento com abordagens não-preemptivas, as tarefas executam até o seu final, sem interrupções, ou são bloqueadas devido a algum acesso exclusivo aos recursos que estão utilizando. O problema que os escalonamentos não-preemptivos apresentam é que em processadores com mais de uma tarefa, pode haver uma demora na inicialização das tarefas, uma vez que devem aguardar o final da execução da tarefa atual.

Para definir a ordem e o tempo de execução das tarefas em cada processador, a presente Tese implementa o algoritmo de escalonamento Preemptivo baseado em Prioridades (PP)

[GOO03]. Este algoritmo prioriza a execução da tarefa com maior prioridade. A definição das prioridades pode ser analisada na Figura 27. Podemos notar que as tarefas que contém prioridades com menores valores são consideradas tarefas de alta prioridade.

Figura 27 - Definição de prioridades de tarefas.

O algoritmo de escalonamento preemptivo baseado em prioridade classifica as tarefas em três estados:

- *Running* (em execução)
  A tarefa está no estado *running* quando o escalonador reserva os recursos de processamento para permitir a sua execução;
- *Waiting* (em espera)
  O estado *waiting* é atribuído quando o sistema operacional bloqueou a tarefa ou a tarefa está esperando devido a uma interrupção da NoC (por exemplo, interrompido por causa da primitiva *receive()* da API de comunicação);
- *Blocked* (bloqueada)
  Quando uma determinada tarefa está pronta para executar e sua prioridade é maior que a tarefa atual em execução, o escalonador coloca a tarefa atual em estado *blocked* e reserva os recursos de processamento para a execução da tarefa com maior prioridade.

O cenário apresentado na Figura 28 mostra duas tarefas mapeadas no mesmo processador. Primeiramente, t_1 inicia seu tempo de computação (C_1). No momento em que a tarefa t_2 estiver pronta para executar no processador (no tempo 2,5 da figura), t_1 entra no estado *blocked* e t_2 inicia seu tempo de computação (C_2). Este bloqueio ocorre devido ao fato de t_2 possuir prioridade maior que t_1 (P_1>P_2), conforme as definições de prioridades citados acima. Aproximadamente no tempo 4,5 da figura, t_2 altera seu estado para *waiting*, permitindo o retorno da execução de t_1. Podemos notar que durante a sequência de execução do cenário, t_1 nunca bloqueia t_2. Sempre que possível t_2 entra em execução e bloqueia apenas quando estiver aguardando por algum dado (através da função bloqueante *receive()*).

Como mencionado no início desta Seção, o problema dos algoritmos de escalonamento preemptivos é que algumas tarefas podem atrasar sua execução e não cumprir com os requisitos de tempo, como podemos notar no tempo 12 da tarefa t_1. A tarefa tem um tempo de computação, para cada iteração, igual a 4. A primeira iteração finaliza juntamente com seu prazo (D_1). Porém, a segunda iteração inicia no tempo 8 e não finaliza sua computação antes de seu prazo finalizar,

dessa forma violando o seu *deadline*. Como t_2 tem a maior prioridade entre as tarefas do processador ela não viola deadlines, pois quando t_2 estiver pronta para executar, os recursos de processamento serão reservados para sua execução.

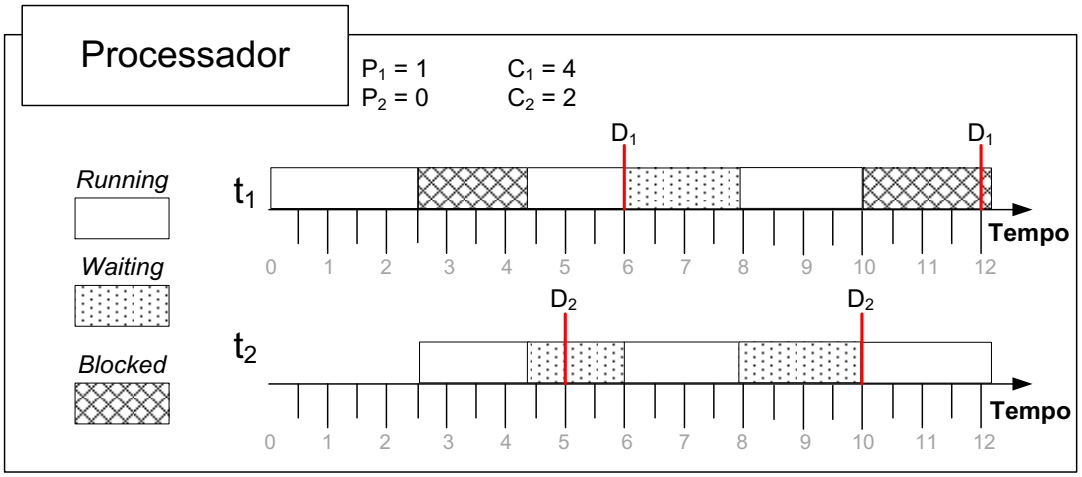


Figura 28 - Exemplo de execução do algoritmo de escalonamento preemptivo baseado em prioridade com duas tarefas compartilhando o mesmo processador. Fonte: [MAD16B]

Para solucionar este problema, o algoritmo de mapeamento deve ter conhecimento dos requisitos de cada tarefa, e avaliar se é possível efetuar o compartilhamento dos recursos de processamento para diferentes tarefas com requisitos de tempo.

5.3 Método de Análise de Escalonabilidade

Esta Seção apresenta a análise de escalonabilidade (do inglês *schedulability*) que é utilizada como função custo do mapeamento de tarefas proposto na Tese. Esta análise tem por objetivo avaliar se um sistema multiprocessado pode garantir o atendimento das restrições de tempo de todas as tarefas a serem executadas pelo sistema [IND14]. A presente Tese implementa a abordagem apresentada por Audsley et. Al. [AUD93], denominada *Response Time Analysis* (RTA). Esta técnica avalia quanto que uma tarefa de maior prioridade, no mesmo processador, pode interferir no atraso do tempo de computação de uma tarefa de menor prioridade.

A Equação 1 calcula o tempo de resposta (R_i) de t_i. O resultado apresentado pela equação apresenta o maior intervalo de tempo entre o início da execução de um *job* de uma tarefa até seu final. O tempo é calculado considerando a interferência de todas as tarefas presentes no processador.

$$R_i = C_i + \sum_{\forall\, t_j \,\epsilon\, hp(t_i)} \left\lceil \frac{R_i}{T_j} \right\rceil . C_j \; + Comm_{\;i}$$

Equação 1 - Equação para definição do *Reponse Time Anaylsis* (RTA). Fonte: [IND14]

A computação da equação 1 ocorre pela adição do tempo de computação requerido por t_i (C_i) em cada iteração, e o tempo de computação de todas as tarefas que podem interromper t_i (C_j)

no mesmo processador. A função $hp(t_i)$ define o conjunto de todas as tarefas, no mesmo processador, com maior prioridade que t_i. Dessa forma, $hp(t_i)$ inclui todas as tarefas (t_j) onde $map(t_i) = map(t_j)$ e $P_i < P_j$. A função $map(t_n)$ retorna a posição de mapeamento de t_n. O custo de comunicação de t_i é representado por $Comm_i$ na equação. Para garantir que a tarefa terá suas restrições de tempo atendidas, a comparação $R_i \leq D_i$ deve ser verdadeira.

A Equação 1 é derivada da Equação 2, de Indrusiak et al. [IND14]. Tal equação não considera que o tempo de comunicação pode ser aplicado no tempo de computação da tarefa. Os testes apresentados na tabela a seguir mostram que a comunicação corresponde a uma parcela muito baixa em comparação com a computação (<6%).

$$R_i = C_i + \sum_{\forall\, t_j \,\epsilon\, hp(t_i)} \left\lceil \frac{R_i}{T_j} \right\rceil . C_j$$

Equação 2 - Equação para definição do *Reponse Time Anaylsis* (RTA). Fonte: [IND14]

O tempo de computação (C_i) inclui o tempo de comunicação de t_i com outras tarefas. A Tabela 12 avalia a relação do tempo de computação sobre o tempo de comunicação para quatro benchmarks, mapeando uma tarefa por PE.

Tabela 12 - Tempo de comunicação e computação para uma iteração de cada becnhmark.

Aplicações	Tempo de Execução (ciclos de relógio)	Tempo de Comunicação (ciclos de relógio)	Computação vs Comunicação
MPEG	70.590	863	1,2%
Dijkstra	78.000	2.517	3,2%
DTW	125.750	1.666	1,3%
VOPD	24.037	1.396	5,8%

Os experimentos demonstraram que o tempo de comunicação é inferior a 6% em relação ao tempo de computação da aplicação para os benchmarks avaliados. Isso ocorre devida a baixa injeção de dados na NoC pelas aplicações. As tarefas de uma aplicação tendem a ser mapeadas na mesma região, minimizando a interferência de outros fluxos de comunicação. Portanto, a sobrecarga de comunicação entre as tarefas comunicantes pode ser incorporada no valor computado de C_i.

A Figura 29 apresenta o uso da Equação 1 para definição do mapeamento de tarefas no sistema, em um exemplo didático. O exemplo demonstrado na figura apresenta a seguinte configuração de MPSoC: instância de tamanho 3x3 com gerenciamento de recursos centralizado (cluster de tamanho 3x3) e 2 tarefas por PE. O conjunto de tarefas utilizado no exemplo é composto de três tarefas (t_a, t_b e t_c) e suas restrições de tempo podem ser analisadas na Figura 29. A tarefa t_a apresenta a maior prioridade entre as tarefas e é a primeira a ser mapeada. Na Etapa 1, faz-se o uso da equação RTA para verificar se a tarefa pode ser mapeada no processador inicial (x,y)=(1,1). A equação retorna para R_a (tempo de resposta) o valor de 4.045 ciclos de relógio. Dessa

forma, torna-se possível o mapeamento de t_a no processador 1x1, pois $R_a \leq D_a$. Na Etapa 2, t_b é a tarefa a ser mapeada. A primeira tentativa de mapeamento ocorre no processador inicial. A tarefa t_a está mapeada no processador em análise e contém uma prioridade maior que t_b. Assim, a equação RTA retorna para R_b o valor de 6.847 ciclos de relógio, indicando que t_a interfere no tempo de resposta de t_b. A tarefa t_b não pode ser mapeada no processador 1x1, pois $R_b > D_b$. O próximo processador a ser analisado para mapear t_b é o (x,y)=(0,1). Nesse caso, o processador não contém tarefas que poderiam interferir o processamento de t_b. Sendo assim, mapeado em 0x1. A Etapa 3 analisa o mapeamento da t_c no processador inicial. O tempo de resposta (R_c) retornado é de 11.803 ciclos de relógio. Como o D_c é 12.000 ciclos de relógio, é possível mapear t_c compartilhando o processador com a tarefa t_a.

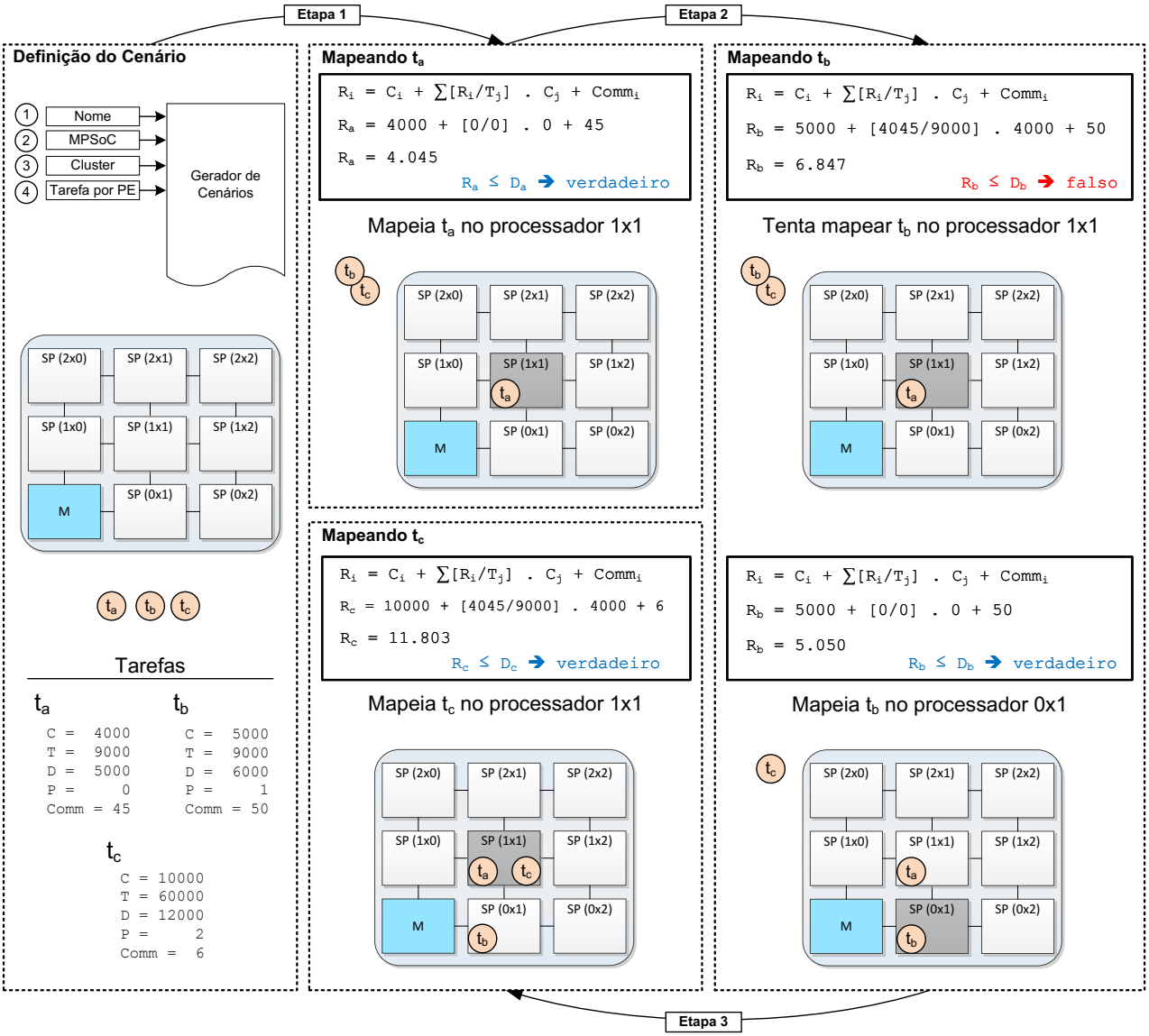


Figura 29 – Utilização da equação RTA para decisão de mapeamento.

5.4 Mapeamento de Tarefas

Esta Seção apresenta o algoritmo de mapeamento de tarefas desenvolvido na presente Tese. O objetivo dos mecanismos de mapeamento é encontrar um processador para que as tarefas possam executar de acordo com uma determinada função custo [SIN13]. Este trabalho implementa uma abordagem de mapeamento de tarefas baseado no modelo RTA.

A heurística de mapeamento utiliza as seguintes definições e funções:

Definição 1 – `N_Apps`

Representa o conjunto de aplicações a serem mapeadas no sistema. Estas aplicações são parametrizadas em tempo de projeto.

Definição 2 – app_i`.size`

Definição da quantidade de tarefas presentes na aplicação app_i, onde $app_i \in N\_Apps$.

Definição 3 – `pe_set(`PE_{add}`,n_hops)`

Esta função retorna um conjunto de PEs que serão analisados para mapear determinada tarefa. A função é parametrizada por PE_{add} (posição do PE atual) e n_hops (distância Manhattan, apresentada na Figura 30, entre o PE_{add} e os PEs a n_hops). O retorno da função é o conjunto de todos os processadores que estiverem a n_hops de PE_{add}. A figura mostra a distância Manhattan de todos os processadores relativos ao PE_{add} (PE azul). Os PEs da cor verde estão a 1 hop de distância de PE_{add}. Os PEs amarelos estão a 2 hops de distância. E assim sucessivamente para os demais PEs.

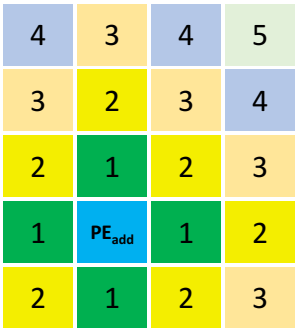


Figura 30 - Definição de região com base na distância de *n_hops* de PE_{add}.

Definição 4 – `rt_available(`pe_i`)`

Definição da função que retorna verdadeiro caso PE_i não tenha tarefas com restrição de tempo-real mapeadas.

Definição 5 – `is_rt_app(`app_i`)`

Definição da função que retorna verdadeiro para se caso app_i for uma aplicação de tarefas com restrição de tempo-real.

Definição 6 – `response_time(`$task_i$`, `pe_i`)`

Representa a função que retorna verdadeiro se o tempo de resposta (R_i da Equação 1) da tarefa $task_i$, mapeado em pe_i, for menor ou igual a seu *deadline* ($R_i \leq D_i$).

O algoritmo proposto busca mapear todas as tarefas de uma determinada aplicação, considerando as aplicações já mapeadas no sistema. Se uma tarefa não pode ser mapeada, isto implica que o conjunto de configurações do MPSoC não atendem às restrições de tempo das tarefas, e a aplicação não pode ser mapeada nas condições atuais do sistema.

O sistema é projetado para dar suporte a aplicações com restrições de tempo-real (RT) e aplicações *Best-Effort* (BE). As aplicações BE não contêm restrições de tempo para avaliar. O pseudocódigo ilustrado na Figura 31 representa o algoritmo de mapeamento. As entradas do algoritmo são: (*i*) aplicação a ser mapeada, app_i e, (*ii*) endereço do PE usado como posição inicial (*seed*) para mapear as tarefas da app_i. O dado de saída do algoritmo é um vetor com a posição de todas as tarefas da app_i. A primeira posição, por padrão, do *seed* para iniciar a execução da heurística de mapeamento é definida com o endereço *(x,y)=(1,1)*. As tarefas são mapeadas, se possível, no processador inicial e nos PEs ao redor dele. A definição do próximo *seed*, para iniciar o mapeamento da próxima aplicação, é a posição do PE com 2 *hops* de distância a partir da tarefa mapeada mais a direita da app_{i-1}, ou a 2 *hops* acima da tarefa mapeada mais a esquerda da app_{i-1}. O processo de eleger o *seed* para mapear uma aplicação garante a localidade das aplicações e minimiza a interferência do fluxo de comunicação entre diferentes aplicações.

```
      Input: application app_i, seed
      Output: app_mapped[app_i.size]

1.   FOR EACH task t_j IN app_i
2.        app_position[t_j] ← -1
3.        n_hops ← 1
4.        WHILE all PEs in the system were not evaluated AND app_position[t_j]=-1 DO
5.             IF is_rt_app(app_i) THEN
6.                  neighbors_list ← pe_set(seed, n_hops)
7.                  FOR EACH PE_k IN neighbors_list
8.                       IF rt_available(PE_k) AND respose_time(t_j, PE_k) THEN
9.                            app_position[t_j] ← PE_k
10.                           break;
11.                      END IF
12.                 END FOR
13.                 IF app_position[t_j] = -1 THEN
14.                      increase(n_hop, 1)
15.                 END IF
16.            ELSIF
17.                 Map the task executing the LEC-DN mapping [MAN15]
18.            END IF
19.       END WHILE
20.       IF app_position[t_j] = -1 THEN
21.            message(Application is not schedulable!)
22.       END IF
23.  END FOR
24.  return app_mapped[]
```

Figura 31 - Pseudo-código do mapeando e tarefas RTA.

O mapeamento inicia com o laço apresentado na linha 1, onde cada tarefa da app_i será mapeada. A linha 2 atribui uma posição inválida para t_j, e a linha 3 define o espaço de busca em torno de t_j a um *hop*. A linha 4 inicia um laço interno, saindo quando todos os PEs do sistema foram avaliados para receber t_j, ou quando t_j foi mapeado. As linhas 5 a 15 mapeiam as tarefas com restrições de tempo-real. A linha 6 faz a busca por todos os PEs candidatos a receber t_j a partir da posição do *seed* e do número de *hops* atuais. O laço representado nas linhas 7 a 12 avalia todos os PEs no conjunto de candidatos, avaliando a disponibilidade do PE e o teste do RTA (linhas 8 a 11). O primeiro PE avaliado, entre os candidatos a receber t_j, que apresentar $R_{task} \leq D_{task}$ é selecionado para receber t_j. As linhas 13 a 15 aumentam o espaço de busca em 1 *hop*, adicionando novos PEs candidatos a receber t_j quando retornar para linha 7. Se a aplicação for BE (linha 16), as tarefas são mapeadas de acordo com a heurística LEC-DN apresentada por Mandelli et. al. [MAN15]. Se alguma determinada tarefa não pode ser mapeada no sistema, então, uma mensagem de aviso é gerada (linha 21) informando que o MPSoC não contém configurações suficientes para executar todas as aplicações garantindo as restrições de tempo-real.

Duas situações podem evitar o mapeamento de determinada aplicação. A primeira é a ausência de PEs disponíveis (situação aplicável para tarefas BE e RT). A segunda situação é quando a função *response_time()* retorna falso para todos os PEs do sistemas (aplicada apenas para as aplicações RT).

O mapeamento apresentado garante que todas as aplicações com restrição de tempo serão atendidas quando executadas no sistema. O projetista configura a plataforma com o conjunto de aplicações (BE e RT) a serem executadas e, também, as configurações do MPSoC (por exemplo, tamanho do MPSoC, tamanho dos *clusters*, tarefas por PE, tamanho da memória, etc.). Dessa forma, se o mapeamento não conseguir achar a posição ideal para alguma tarefa, o projetista deve modificar as características do MPSoC ou rever os requisitos das aplicações a serem executadas no sistema.

Esta etapa do projeto assume um mapeamento estático (as decisões de mapeamento são feitas em tempo de projeto). Porém, o mapeamento proposto pode ser adaptado para executar em tempo de simulação. Um PE gerente com conhecimento da utilização do sistema pode mapear as aplicações RT de acordo com a heurística apresentada no pseudocódigo.

A Figura 32 apresenta o fluxo de projeto desde a configuração do sistema até sua simulação. Esta Figura representa as ações de projeto com base no pseudocódigo. Primeiramente, a etapa 1 configura o sistema e suas aplicações. Na etapa 2 o algoritmo de mapeamento com base no RTA é executado, e se todas as tarefas (etapa 3) puderem ser mapeadas, a execução é iniciada (etapa 4). Caso contrário, alterações no sistema e/ou nas aplicações devem ser consideradas.

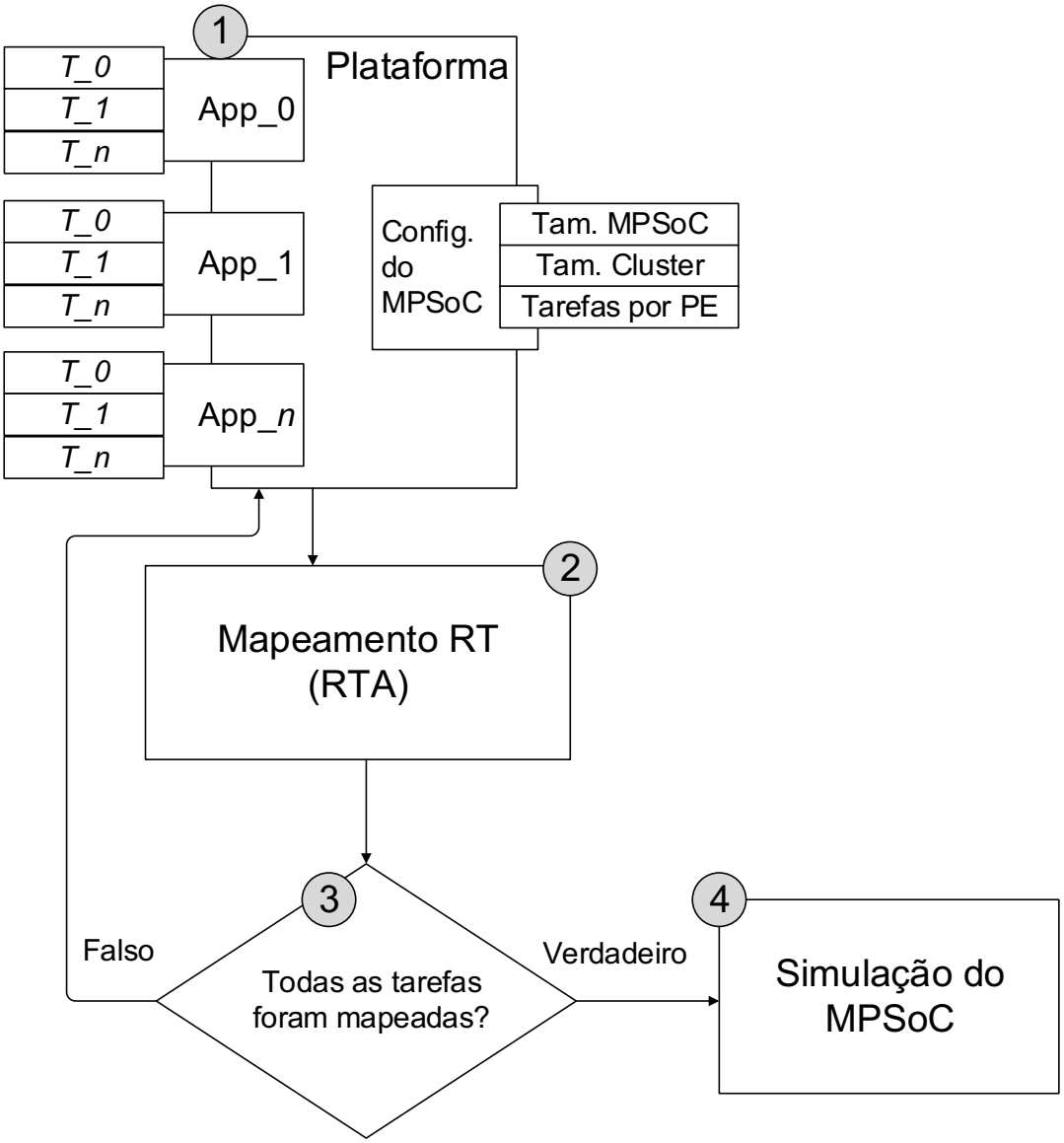


Figura 32 - Fluxo de projeto basedo nas informações de mapeamento.

6 RESULTADOS

Este Capítulo apresenta os resultados da implementação da arquitetura HeMPS com as técnicas de escalonamento e mapeamento de tarefas para sistemas com restrições de tempo-real, apresentadas no Capítulo 5. Para efetuar a avaliação da plataforma, os cenários de teste são descritos na Seção 6.1. Os mecanismos de mapeamento e escalonamento são avaliados nas Seções 6.2 e 6.3, respectivamente.

Os cenários consistem em um conjunto de processadores homogêneos conectados através de uma NoC com chaveamento por pacotes, sem a utilização de canais virtuais. Para reduzir o tráfego de dados na rede os cenários contêm apenas memórias locais, usando o modelo de comunicação por troca de mensagens.

6.1 Descrição dos Cenários Avaliados

Os cenários avaliados na presente Tese são compostos por quatro das aplicações apresentadas na Seção 4.1: (1) MPEG; (2) Dijkstra; (3) DTW; e (4) VOPD. A ocupação do MPSoC refere-se ao número de tarefas que uma determinada configuração de MPSoC pode executar simultaneamente. Essa ocupação é de acordo com o número de PEs e o número de tarefas que cada PE pode executar simultaneamente. Por exemplo, um MPSoC de tamanho 6x6 com um PE gerente, podendo executar duas tarefas por PE, pode mapear 70 tarefas simultaneamente no sistema (35 SPs executando 2 tarefas).

Os experimentos realizados na Tese adotam dois tamanhos de MPSoC: (*i*) 6x6; e (*ii*) 8x8. Cada PE pode executar de uma a três tarefas simultaneamente. A ocupação do MPSoC varia entre 10% e 100%. Para automatizar a geração dos cenários de teste, foi desenvolvido um *script* que analisa o tamanho do MPSoC e configura os cenários selecionando automaticamente o conjunto de aplicações que se enquadram nas definições do sistema.

A Tabela 13 apresenta o número de *deadlines* que o sistema deve atender, assumindo que todas as aplicações do sistema contêm restrições de tempo-real (cenários com 100% de aplicações de tempo-real). Quando o método analítico RTA (função custo do mapeamento) não consegue definir a posição para o mapeamento de todas as tarefas, em uma determinada configuração de MPSoC, o número de *deadlines* não é apresentado na tabela. Note que como o número de tarefas por PE aumenta, a ocupação por cenário diminui. A razão para que isso aconteça é que o mapeamento de tarefas RT começa a não mapear tarefas com maior compartilhamento de CPU. A quantidade de *deadlines* que o sistema deve atender varia conforme as aplicações que compõe o cenário.

A Tabela 14 apresenta um cenário contendo uma mistura de aplicações de tempo-real e aplicações *best-effort*. Para estes cenários somente a aplicação MPEG contém restrições de tempo-real. O objetivo dos cenários com 100% de aplicações de tempo-real é analisar qual o percentual

de ocupação que o MPSoC atenderia às restrições de tempo. Para os testes de RT com BE, a ocupação do MPSoC é maior, atingindo até 60% no MPSoC de tamanho 8x8, com três tarefas por PE.

Tabela 13 – Número de *deadlines* em cenários 100% de tempo-real (38 cenários).

Ocupação do MPSoC (%)	MPSoC 6x6			MPSoC 8x8		
	Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3
0	100	120	161	100	161	112
20	120	281	173	161	273	513
30	161	173	393	112	513	686
40	281	293	554	273	566	-
50	281	393	-	393	786	-
60	173	554	-	513	-	-
70	173	-	-	405	-	-
80	293	-	-	566	-	-
90	393	-	-	686	-	-
100	393	-	-	786	-	-

Tabela 14 – Número de *deadlines* em cenários com aplicações RT e BE (47 cenários).

Ocupação do MPSoC (%)	MPSoC 6x6			MPSoC 8x8		
	Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3
10	100	100	100	100	100	100
20	100	100	100	100	100	200
30	100	100	100	100	200	300
40	100	100	200	100	300	400
50	100	200	300	100	300	500
60	100	200	300	200	400	600
70	100	200	-	200	500	-
80	100	300	-	300	-	-
90	200	-	-	300	-	-
100	200	-	-	300	-	-

6.2 Resultado dos Testes de Mapeamento

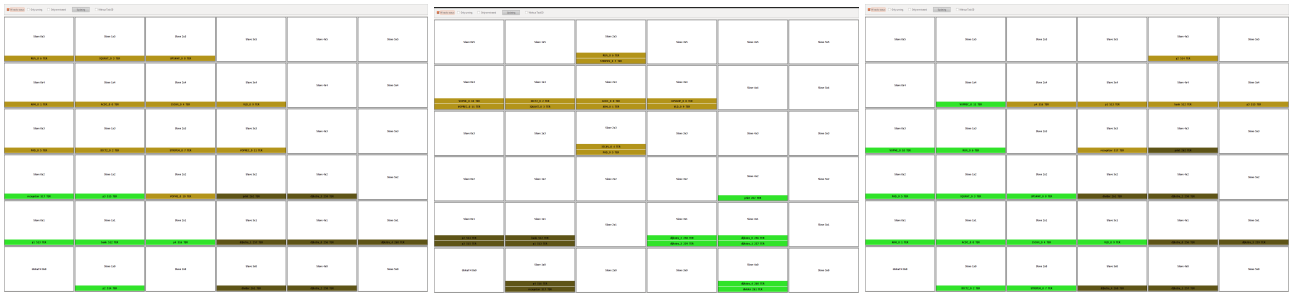
Nesta Seção apresentam-se os resultados de mapeamento de tarefas analisados com base em dois algoritmos de mapeamento como referência para os testes: (*i*) *Low Energy Consumption – Dependence Neighborhood* (LEC-DN) [MAN15]; e (*ii*) *Hierarchical Energy-Aware Task Mapping* (HEAT) [CAS16].

Mandelli et. al. [MAN15] apresentam o mapeamento LEC-DN que usa a energia total da comunicação entre tarefas como função custo do mapeamento. Castilhos et. al. [CAS16] apresentam o HEAT, uma proposta de mapeamento dinâmico que faz uma mescla entre a carga de trabalho dos processadores e o volume de comunicação da NoC. O principal objetivo do HEAT é diminuir a ocorrência de *hotspots*, dessa maneira aumentando o tempo de vida do sistema.

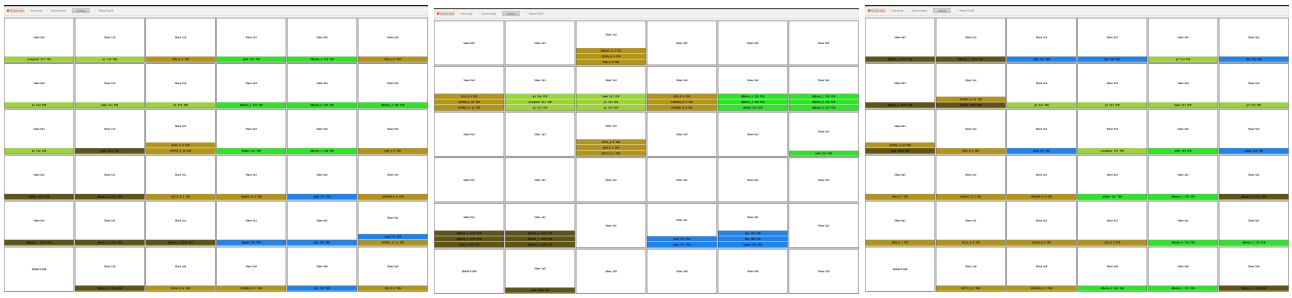
Considerando-se os cenários apresentados nas tabelas da Seção anterior, a primeira avaliação baseia-se nos cenários com 100% de aplicações com restrições de tempo-real. A Figura

33 apresenta os mapeamentos das tarefas para MPSoCs com configuração de tamanho 6x6 com 2 (Figura 33(a)) e 3 (Figura 33(b)) páginas por PE, com ocupação próxima a 40%. A figura apresenta o mapeamento das tarefas nas diferentes heurísticas de mapeamento.

As figuras de mapeamento apresentadas a seguir formam geradas a partir de um framework para depuração de MPSoCs [RUA14]. Cada quadrado apresentado na figura representa um PE. Os retângulos coloridos representam tarefas de aplicações (conjunto de retângulos com cores iguais). Quando um PE compartilha seus recursos entre um ou mais tarefas, retângulos coloridos são superpostos dentro do quadrado (PE).



(a) HEAT, LEC-DN e RTA em MPSoC 6x6 com 2 páginas por PE e 40% de ocupação



(b) HEAT, LEC-DN e RTA em MPSoC 6x6 com 3 páginas por PE e 40% de ocupação

Figura 33 - Mapeamento de tarefas para cenários 100% real time.

Pode-se notar que o mapeamento LEC-DN propõe um posicionamento próximo entre as tarefas comunicantes, em muitos casos fazendo-as compartilhar o mesmo PE. Por outro lado, o mapeamento HEAT trabalha com base na distribuição da carga de trabalho no MPSoC. O algoritmo proposto nesta Tese compartilha o PE entre tarefas apenas quando uma tarefa não interfere na computação de outra. Dessa maneira, o mapeamento baseado no RTA tende a distribuir a carga de trabalho no sistema, mas sempre testando a possibilidade de compartilhamento de processamento.

Na Tabela 15 apresentam-se os percentuais de *deadlines* atendidos dos cenários com todas as aplicações com restrição de tempo-real, nas configurações de MPSoC com tamanho 6x6. Podemos notar que a heurística de mapeamento baseada no modelo analítico RTA e o mapeamento HEAT apresentam um atendimento total das restrições de tempo-real das aplicações do sistema, para esta configuração de MPSoC. Por outro lado, o mapeamento LEC-DN por reduzir a energia de comunicação não atende a todos os *deadlines* do sistema. Em alguns casos o

mapeamento LEC-DN fica abaixo de 60% de *deadlines* atendidos.

Tabela 15 – Percentual de *deadlines* atendidos para os cenários 100% RT.

Ocupação do MPSoC (%)	MPSoC de tamanho 6x6								
	RTA			HEAT			LEC-DN		
	Tarefas por PE			Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3	1	2	3
10	100%	100%	100%	100%	100%	100%	100%	54%	23%
20	100%	100%	100%	100%	100%	100%	100%	100%	28%
30	100%	100%	100%	100%	100%	100%	100%	33%	41%
40	100%	100%	100%	100%	100%	100%	100%	50%	36%
50	100%	100%	-	100%	100%	-	100%	58%	-
60	100%	100%	-	100%	100%	-	100%	50%	-
70	100%	-	-	100%	-	-	100%	-	-
80	100%	-	-	100%	-	-	100%	-	-
90	100%	-	-	100%	-	-	100%	-	-
100	100%	-	-	100%	-	-	100%	-	-

A Tabela 16 apresenta a análise dos resultados obtidos através da configuração de MPSoCs com tamanho 8x8. Da mesma maneira que os dados analisados na tabela anterior, para MPSoCs com esta configuração os mapeamentos HEAT e RTA se sobressaem ao LEC-DN, que novamente atinge, em alguns casos, menos de 60% de garantia no atendimento de *deadlines*.

Tabela 16 – Percentual de *deadlines* atendidos para os cenários 100% RT.

Ocupação do MPSoC (%)	MPSoC de tamanho 8x8								
	RTA			HEAT			LEC-DN		
	Tarefas por PE			Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3	1	2	3
10	100%	100%	100%	100%	100%	100%	100%	47%	63%
20	100%	100%	100%	100%	100%	100%	100%	60%	42%
30	100%	100%	100%	100%	100%	100%	100%	57%	38%
40	100%	100%	-	100%	100%	-	100%	51%	-
50	100%	100%	-	100%	100%	-	100%	54%	-
60	100%	-	-	100%	-	-	100%	-	-
70	100%	-	-	100%	-	-	100%	-	-
80	100%	-	-	100%	-	-	100%	-	-
90	100%	-	-	100%	-	-	100%	-	-
100	100%	-	-	100%	-	-	100%	-	-

Em todos os cenários avaliados, os mapeamentos HEAT e RTA atendem a todas as restrições de tempo-real estabelecidas pelas aplicações. Nos MPSoCs de tamanho 6x6 e 8x8, com mais de uma tarefa por PE, o mapeamento LEC-DN atingiu uma perda entre 60 e 80% dos *deadlines* nos cenários apresentados. A Figura 34 apresenta o percentual de *deadlines* atendidos por cada mapeamento. MPSoCs configurados com baixa taxa de ocupação favorecem o mapeamento HEAT, que minimiza o compartilhamento de PEs distribuindo a carga de trabalho pelo sistema. Por outro lado, o mapeamento LEC-DN minimiza a energia de comunicação, favorecendo o compartilhamento de PEs, e consequentemente induzindo a violações de *deadlines*.

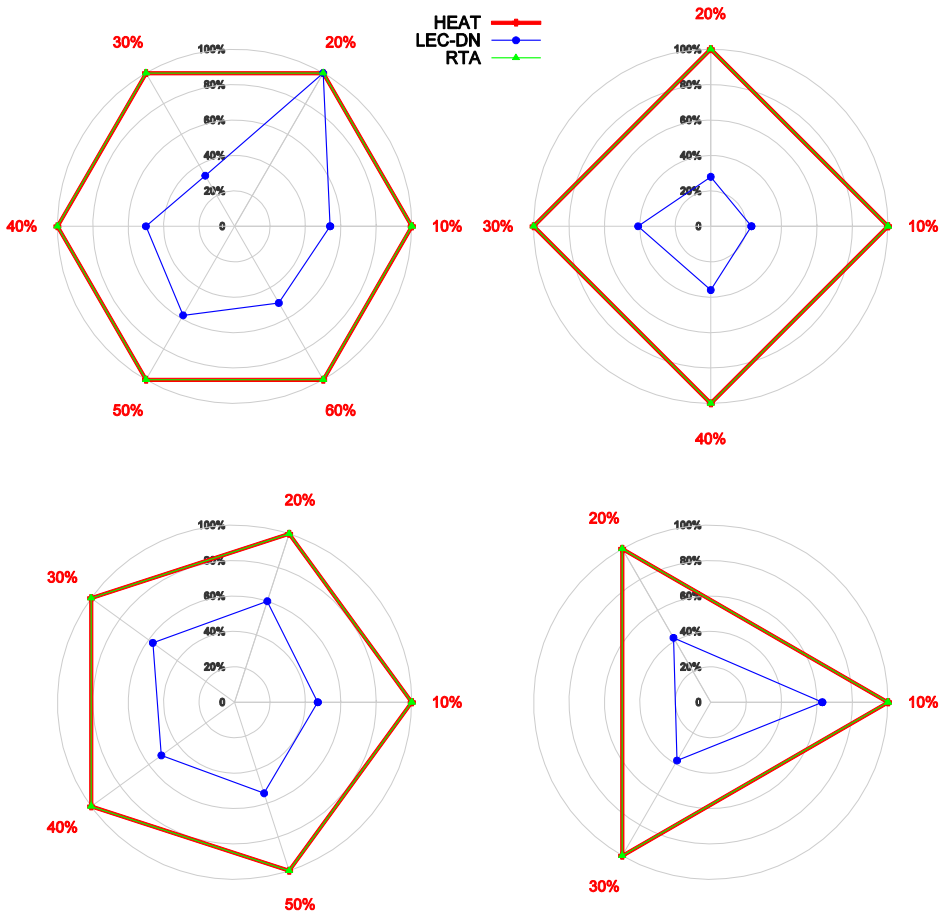


Figura 34 - Percentual de *deadlines* atendidos por cada heurística de mapeamento (números externos: ocupação do MPSoC; números verticais: *deadlines* atendidos). Fonte: [MAD16B]

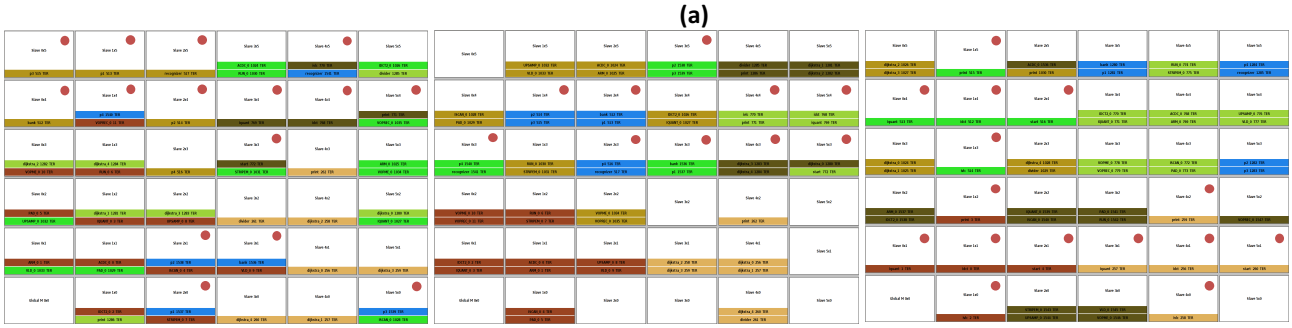
A segunda avaliação dos testes de mapeamento baseia-se nos cenários com uma mistura de aplicações com restrições de tempo-real e aplicações *best-effort* (BE). As aplicações BE não apresentam restrições de tempo-real. Assim, o resultado da computação deste tipo de aplicação não depende do momento em que foi gerado. A Figura 35(a) apresenta o resultado dos mapeamentos das tarefas para MPSoCs com configuração de tamanho 6x6 com 1 página por PE, com ocupação próxima a 100%. Para esta configuração de cenário as aplicações são distribuídas no sistema, sem compartilhamento de PE.

Na Figura 35(b) apresenta-se o mapa das tarefas na configuração de MPSoCs de tamanho 6x6 com 2 tarefas por PE, com 80% de ocupação do MPSoC. Podemos notar que o mapeamento LEC-DN minimiza o gasto energético de comunicação compartilhando PEs entre tarefas comunicantes. Por outro lado, o mapeamento HEAT compartilha PEs distribuindo tarefas com baixa carga de trabalho no mesmo PE, enquanto as tarefas de mais alta carga de computação não fazem compartilhamento de processador. O mapeamento RTA compartilha PEs apenas para as aplicações BE, uma vez que as aplicações RT, selecionadas para os testes, não podem fazer compartilhamento de PE com as restrições estabelecidas pela aplicação. Os PEs que executam tarefas com restrições de tempo-real estão marcados com um círculo vermelho.

A Figura 35(c) apresenta o mapeamento de tarefas em MPSoC de tamanho 6x6 com 3 páginas por PE, com 60% de ocupação do MPSoC. Pode-se notar que os mapeamentos se tornaram similares aos apresentados na figura anterior. O mapeamento LEC-DN têm seus resultados penalizados pelo alto compartilhamento de PEs. O HEAT distribui a carga das aplicações, em alguns casos compartilhando PEs. Já o mapeamento RTA distingue as aplicações não compartilhando as tarefas com restrições de tempo-real.

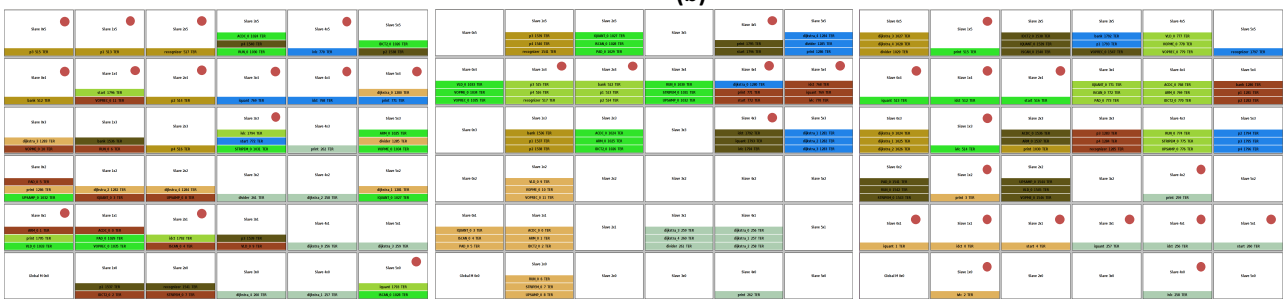
HEAT LEC-DN RTA

(a)



HEAT LEC-DN RTA

(b)



HEAT LEC-DN RTA

(c)

Figura 35 - Mapeamento de tarefas em MPSoC de tamanho 6x6: (a) 1 página por PE com ocupação próxima a 100%; (b) 2 páginas por PE, com 80% de ocupação; (c) 3 páginas por PE, com 60% de ocupação. Círculos vermelhos indicam tarefas tempo-real.

Pode-se notar que as heurísticas de mapeamento LEC-DN e HEAT mapeiam as tarefas sem distinção do tipo de aplicação. Por outro lado, a heurística de mapeamento baseada no RTA distingue as aplicações RT das BE.

Na Tabela 17 apresentam-se os percentuais dos *deadlines* atendidos para os cenários com aplicações RT e BE, na configuração de MPSoCs com tamanho 6x6. Podemos notar que o mapeamento RTA atende a todos os prazos estabelecidos pelas aplicações RT. O mapeamento LEC-

DN não atende às restrições de tempo das aplicações, nas configurações de MPSoC com múltiplas tarefas executando no mesmo processador. A heurística de mapeamento HEAT começa a apresentar violações de *deadlines* quando a ocupação do MPSoC aumenta, e para cada cenário com duas ou mais tarefas por PE.

Tabela 17 – Percentual de *deadlines* atendidos para os cenários com aplicações RT e BE.

Ocupação do MPSoC (%)	MPSoC de tamanho 6x6								
	RTA			HEAT			LEC-DN		
	Tarefas por PE			Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3	1	2	3
10	100%	100%	100%	100%	100%	100%	100%	83%	63%
20	100%	100%	100%	100%	100%	100%	100%	83%	59%
30	100%	100%	100%	100%	100%	100%	100%	83%	59%
40	100%	100%	100%	100%	100%	100%	100%	83%	59%
50	100%	100%	100%	100%	100%	91%	100%	83%	67%
60	100%	100%	100%	100%	100%	94%	100%	84%	67%
70	100%	100%	-	100%	95%	-	100%	93%	-
80	100%	100%	-	100%	97%	-	100%	89%	-
90	100%	-	-	100%	-	-	100%	-	-
100	100%	-	-	100%	-	-	100%	-	-

A Tabela 18 apresenta os dados coletados nas configurações de MPSoCs com tamanho 8x8. Nota-se que a amostra dos dados é similar às apresentadas na tabela anterior. O mapeamento RTA não viola *deadlines* em nenhum cenário de teste. Por outro lado, o mapeamento HEAT, em alguns casos, fica abaixo de 90% de atendimento, tornando o mapeamento inviável para sistemas *hard-RT*. O mapeamento LEC-DN mantém suas características de compartilhamento de PE, não atendendo aos prazos estabelecidos pela aplicação.

Tabela 18 – Percentual de *deadlines* atendidos para os cenários com aplicações RT e BE.

Ocupação do MPSoC (%)	MPSoC de tamanho 8x8								
	RTA			HEAT			LEC-DN		
	Tarefas por PE			Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3	1	2	3
10	100%	100%	100%	100%	100%	100%	100%	83%	62%
20	100%	100%	100%	100%	100%	100%	100%	83%	59%
30	100%	100%	100%	100%	100%	100%	100%	83%	59%
40	100%	100%	100%	100%	100%	99%	100%	83%	62%
50	100%	100%	100%	100%	100%	88%	100%	83%	68%
60	100%	100%	100%	100%	99%	88%	100%	86%	70%
70	100%	100%	-	100%	96%	-	100%	90%	-
80	100%	-	-	100%	-	-	100%	-	-
90	100%	-	-	100%	-	-	100%	-	-
100	100%	-	-	100%	-	-	100%	-	-

Para os cenários com 2 ou 3 tarefas por PE, podemos analisar que apenas o mapeamento RTA atendeu a todos os prazos em todos os cenários. A Figura 36 apresenta o percentual de *deadlines* atendidos em cada mapeamento. Os números externos (em vermelho) representam a

ocupação do MPSoC, definindo a quantidade de tarefas a serem executadas no sistema. Os números na vertical correspondem ao percentual de *deadlines* que foram atendidos no cenário. As linhas representam a heurística de mapeamento, conforme definido na parte superior da figura.

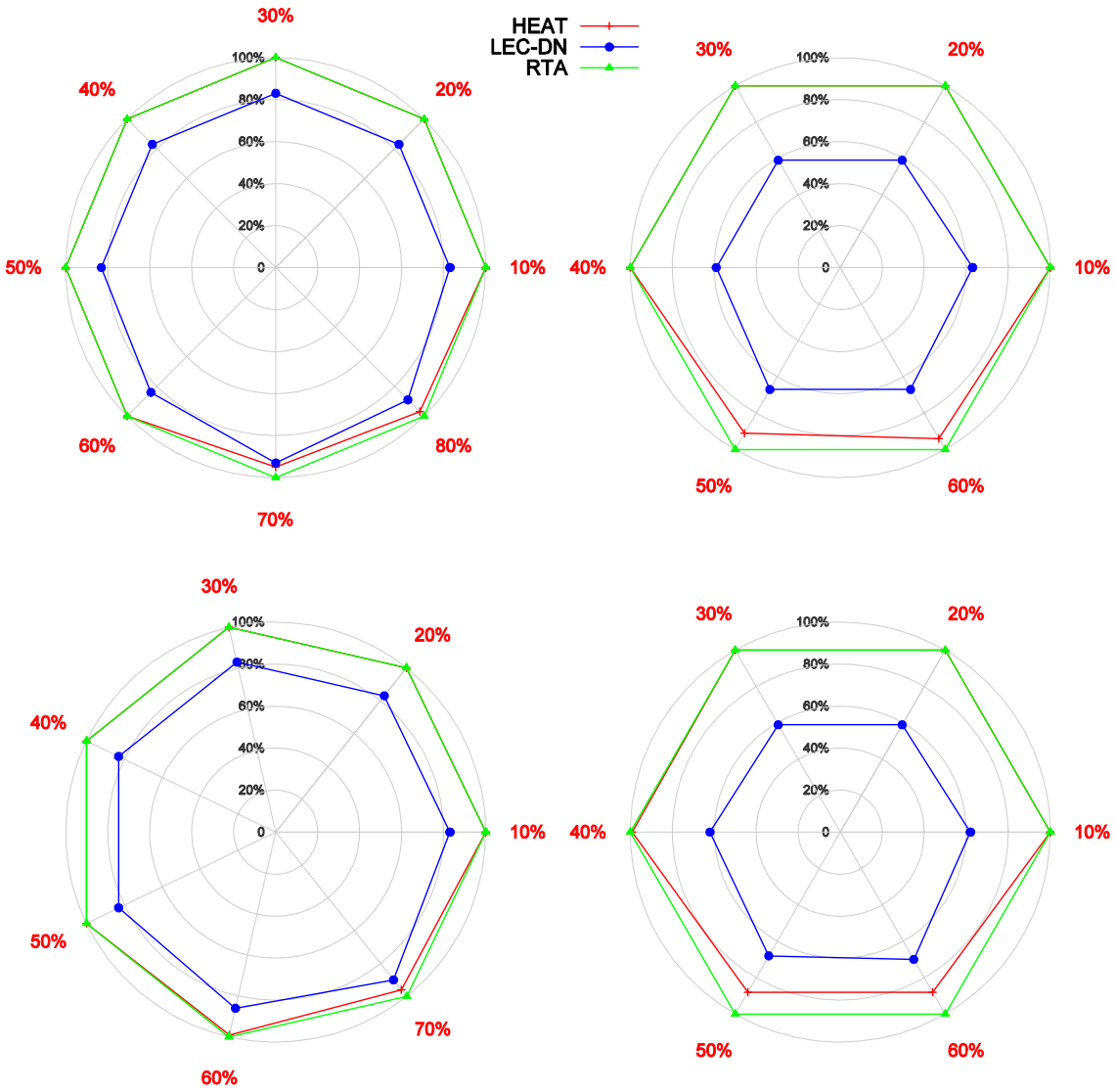


Figura 36 – Percentual de *deadlines* atendidos por cada heurística de mapeamento (números externos: ocupação do MPSoC; números verticais: *deadlines* atendidos). Fonte: [MAD16B]

Nos cenários que apresentam uma mescla de aplicações RT e BE, o comportamento dos dados analisados é alterado comparado com os cenários compostos apenas de aplicações com restrições de tempo-real. O mapeamento LEC-DN sempre viola as restrições de tempo. Tais resultados demonstram que esta heurística não deve ser aplicada em cenários com aplicações de tempo-real. O mapeamento HEAT, nos cenários com duas e três tarefas por PE, atende 96% e 94% dos *deadlines*, respectivamente. Como o objetivo da heurística é otimizar a carga de trabalho evitando *hotspots* no sistema, a interferência entre as tarefas com restrições de tempo no mesmo

processador não é levada em consideração, resultando na violação de *deadlines*.

Pode-se notar que o mapeamento baseado no método analítico RTA garante 100% dos *deadlines* em todos os cenários apresentados. Tais resultados mostraram que uma função de otimização explícita deve ser adotada na heurística de mapeamento, garantindo que os dados resultantes das aplicações RT aconteçam no tempo estabelecido pela aplicação. Funções de otimização secundárias podem ser adicionadas ao mapeamento RTA, avaliando, por exemplo, a presença de *hotspots* no sistema.

6.3 Resultado dos Testes de Escalonamento

Nesta Seção apresentam-se os resultados obtidos nos testes de escalonamento de tarefas. Para avaliar o escalonador desenvolvido nesta Tese, escalonador preemptivo por prioridade (PP), foi utilizado o mecanismo de escalonamento *Least Slack Time* (LST) [RUA15], como referência para os testes.

No escalonador LST as tarefas com restrições de tempo-real têm prioridades maiores que as tarefas BE. As tarefas RT são escalonadas de acordo com seus *slacks time* (tempos de folga). Se duas ou mais tarefas RT apresentarem o mesmo tempo de folga, então um algoritmo Round-Robin é usado para selecionar a próxima tarefa a ser escalonada. O Round-Robin é implementado, também, para escalonar tarefas BE se o processador não conter tarefas RT alocadas ou se todas as tarefas RT estiverem em estado de espera. Em [LI03], os autores apresentam o algoritmo *Round-Robin*. Esse algoritmo garante que cada tarefa será alocada no processador por um determinado intervalo de tempo fixo, denominado *time-slice*. Para cada ciclo de relógio, um contador de *time-slice* é incrementado. Quando uma tarefa completa seu *time-slice*, o seu contador é zerado e a próxima tarefa da fila é escalonada, passando a ser executada com o mesmo período de tempo.

Na Tabela 19 apresentam-se os percentuais de *deadlines* atendidos, por cada escalonador, dos cenários com todas as aplicações com restrição de tempo-real, nas configurações de MPSoC com tamanho 6x6 e 8x8. Podemos notar que o mecanismo de escalonamento preemptivo com prioridades (PP) atende a todas as restrições de tempo estabelecidas pelas aplicações. Por outro lado, podemos notar que o escalonador LST não atende as restrições de tempo quando a quantidade de tarefas por PEs e a ocupação do MPSoC aumentam. Isso ocorre por dois motivos: (*i*) a heurística do LST processa a cada 16k de ciclos de relógio, afetando o processamento da tarefa; (*ii*) para os casos em que há compartilhamento de PE, o LST escalona as tarefas sem considerar as prioridades das tarefas com restrições de tempo-real.

A Tabela 20 apresenta os dados obtidos nos cenários com a mistura de aplicações RT e BE. Pode-se notar que o LST, semelhante aos dados apresentados na tabela anterior, viola *deadlines* das aplicações. Ao contrário do escalonador PP que, por considerar as prioridades das tarefas, garante o atendimento às restrições das aplicações.

Tabela 19 – Percentual de *deadlines* atendidos para os cenários 100% RT.

Ocupação do MPSoC (%)	MPSoC de tamanho 6x6						MPSoC de tamanho 8x8					
	PP			LST			PP			LST		
	Tarefas por PE			Tarefas por PE			Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3	1	2	3	1	2	3
10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
20	100%	100%	100%	100%	99%	99%	100%	100%	100%	100%	100%	99%
30	100%	100%	100%	100%	99%	99%	100%	100%	100%	100%	99%	99%
40	100%	100%	100%	100%	98%	97%	100%	100%	-	100%	99%	-
50	100%	100%	-	100%	96%	-	100%	100%	-	100%	96%	-
60	100%	100%	-	100%	96%	-	100%	-	-	100%	-	-
70	100%	-	-	100%	-	-	100%	-	-	100%	-	-
80	100%	-	-	100%	-	-	100%	-	-	100%	-	-
90	100%	-	-	100%	-	-	100%	-	-	100%	-	-
100	100%	-	-	100%	-	-	100%	-	-	100%	-	-

Tabela 20 – Percentual de *deadlines* atendidos para os cenários RT e BE.

Ocupação do MPSoC (%)	MPSoC de tamanho 6x6						MPSoC de tamanho 8x8					
	PP			LST			PP			LST		
	Tarefas por PE			Tarefas por PE			Tarefas por PE			Tarefas por PE		
	1	2	3	1	2	3	1	2	3	1	2	3
10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
20	100%	100%	100%	100%	81%	79%	100%	100%	100%	100%	82%	89%
30	100%	100%	100%	100%	81%	58%	100%	100%	100%	100%	86%	89%
40	100%	100%	100%	100%	81%	79%	100%	100%	100%	100%	85%	81%
50	100%	100%	100%	100%	81%	78%	100%	100%	100%	100%	84%	79%
60	100%	100%	100%	100%	80%	78%	100%	100%	100%	100%	81%	78%
70	100%	100%	-	100%	75%	-	100%	100%	-	100%	82%	-
80	100%	100%	-	100%	83%	-	100%	-	-	100%	-	-
90	100%	-	-	100%	-	-	100%	-	-	100%	-	-
100	100%	-	-	100%	-	-	100%	-	-	100%	-	-

Com base nos testes apresentados nesta Seção, podemos concluir que o PP mostrou ser o escalonador indicado para sistemas com restrições *hard real-time*. Todos os testes apresentados na Seção 6.2 foram realizados utilizando o escalonador PP. Desta maneira, o escalonar PP e o mapeamento RTA são as heurísticas adequadas para este tipo de sistema. Por outro lado, o escalonador LST pode ser utilizado em sistemas com restrições *soft real-time*, onde em alguns momentos as violações são aceitáveis.

6.4 Análise de Latência de Comunicação

Esta Seção apresenta uma análise da latência média de comunicação entre as tarefas. Os testes foram feitos com base em duas aplicações: (*i*) MPEG, aplicação com alto volume de comunicação; (*ii*) Dijkstra, aplicação onde o tempo de processamento é dominante em relação ao tempo de comunicação. As Seções 6.2 e 6.3 avaliaram o aspecto computação. A presente seção realiza a análise da comunicação. Esta análise demonstra que as aplicações com restrições de

tempo-real mantêm uma latência com baixo *jitter*, fazendo com que as tarefas RT não fiquem bloqueadas aguardando a produção de dados.

Assim como os experimentos anteriores, os cenários configurados para análise de latência de comunicação adotam dois tamanhos de MPSoC: (*i*) 6x6; e (*ii*) 8x8. Todos os PEs podem executar de uma a três tarefas simultaneamente. Utilizou-se o mapeamento RTA e o escalonador PP para a realização dos testes. O primeiro teste realizado foi em uma aplicação com característica de alto volume de comunicação.

Os primeiros cenários apresentados referem-se à MPSoCs com tamanho 6x6. A Figura 37 apresenta o gráfico com os dados das latências de comunicação de uma aplicação MPEG, executando-a em um MPSoC com 1 tarefa por PE e 60% de ocupação. A latência média ficou em 40.800 ciclos de relógio. Na Figura 38, apresenta-se o gráfico de latência média de comunicação de três aplicações MPEG, executando-as em um MPSoC com 3 tarefas por PE e 50% de ocupação. Pode-se notar que a latência média de comunicação da aplicação constantemente varia. Em ambos os gráficos apresentados, a latência média ficou em 41.400 ciclos de relógio para todas as aplicações analisadas.

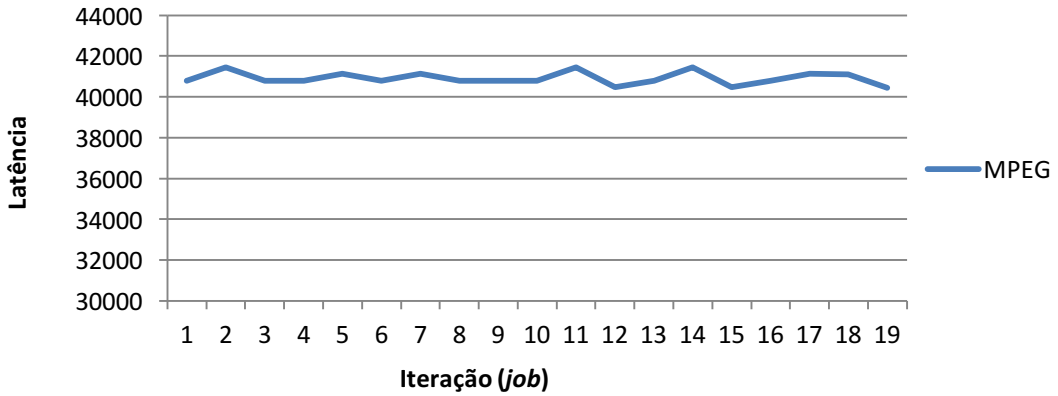


Figura 37 – Variação de latência de comunicação de uma aplicação RT em MPSoC 6x6 com 1 tarefa por PE e 60% de ocupação.

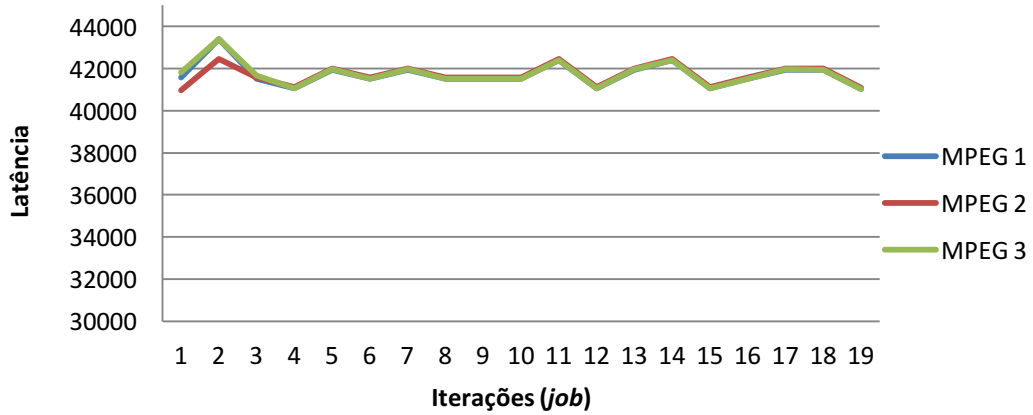


Figura 38 – Variação de latência de comunicação de três aplicações RT em MPSoC 6x6 com 3 tarefas por PE e 50% de ocupação.

Os próximos cenários são executados em MPSoCs de tamanho 8x8. Assim como apresentado nos casos de teste para MPSoCs com tamanho 6x6, as latências médias de comunicação analisadas nestes cenários ficou em 41.500 ciclos de relógio. Pode-se analisar na Figura 39 e na Figura 40 que as variações da aplicação MPEG mantêm a mesma latência, independente do tamanho do MPSoC, quantidade de tarefas ou carga de ocupação.

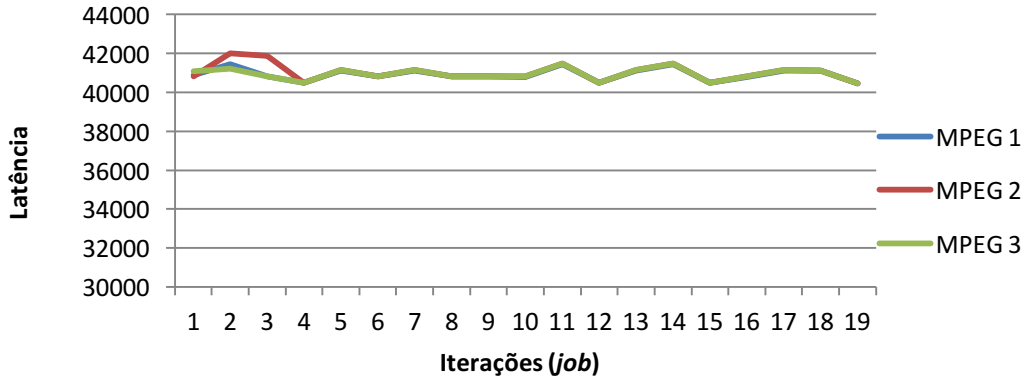


Figura 39 – Variação de latência de comunicação de três aplicações RT em MPSoC 8x8 com 1 tarefa por PE e 80% de ocupação.

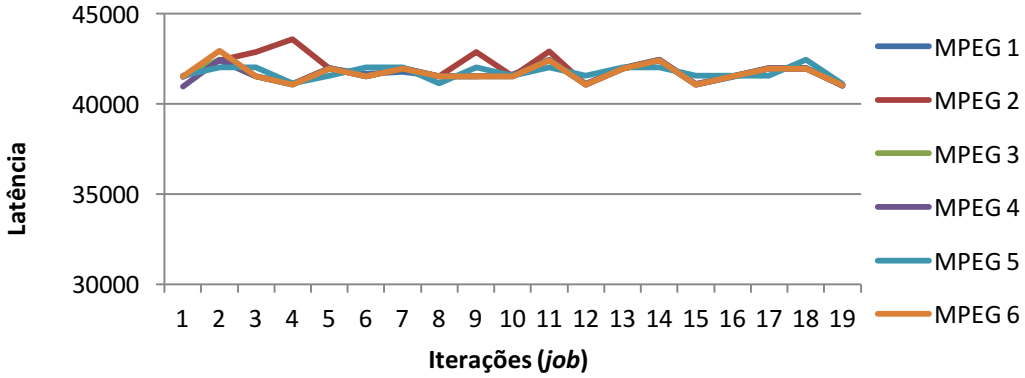


Figura 40 – Variação de latência de comunicação de seis aplicações RT em MPSoC 8x8 com 3 tarefas por PE e 60% de ocupação.

Os casos avaliados anteriormente são realizados com uma aplicação com características de alto volume de comunicação. Para analisar aplicações com característica onde a computação predomina sobre a comunicação, os mesmos testes foram efetuados substituindo as aplicações MPEG por aplicações Dijkstra com restrições de tempo-real. Primeiramente, analisou-se a variação da latência de comunicação de uma e duas aplicações RT, executando-as em MPSoCs de tamanho 6x6 com 1 e 3 páginas por PE, respectivamente. A Figura 41 e a Figura 42 apresentam uma latência média de comunicação em 12.200 e 13.400 ciclos de relógio, respectivamente, para a aplicação Dijkstra.

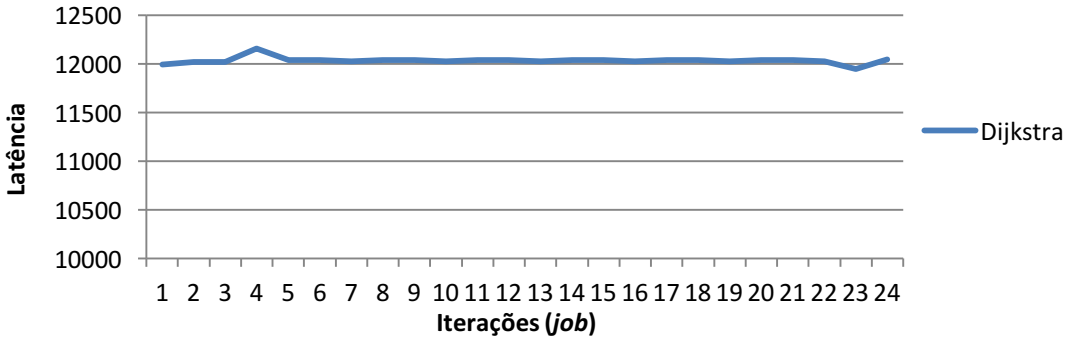


Figura 41 – Variação de latência de comunicação de uma aplicação Dijkstra em MPSoC 6x6 com 1 tarefa por PE e 80% de ocupação.

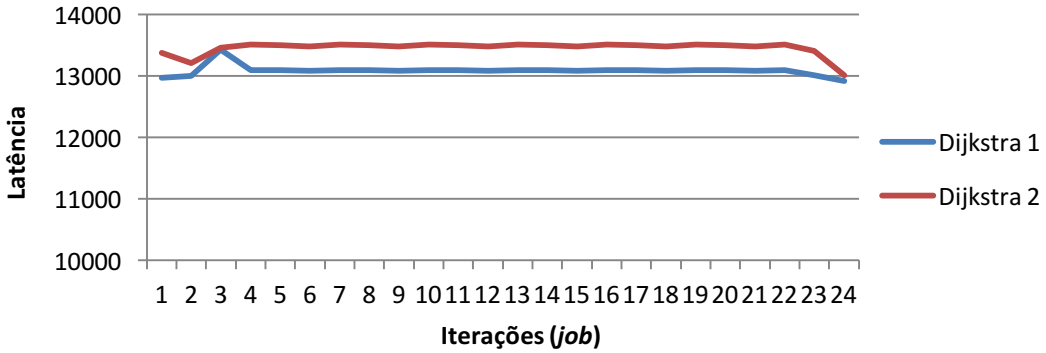


Figura 42 – Variação de latência de comunicação de duas aplicações Dijkstra em MPSoC 6x6 com 3 tarefas por PE e 50% de ocupação.

Os gráficos apresentados na Figura 43 e na Figura 44 representam os dados coletados dos testes da execução de duas e quatro aplicações Dijkstra em MPSoCs de tamanho 8x8 com 1 e 2 tarefas por PE, respectivamente. Pode-se analisar que a latência média de comunicação, assim como nos testes analisados em MPSoCs de tamanho 6x6, atingiu os valores entre 12.200 e 13.400 ciclos de relógio, respectivamente.

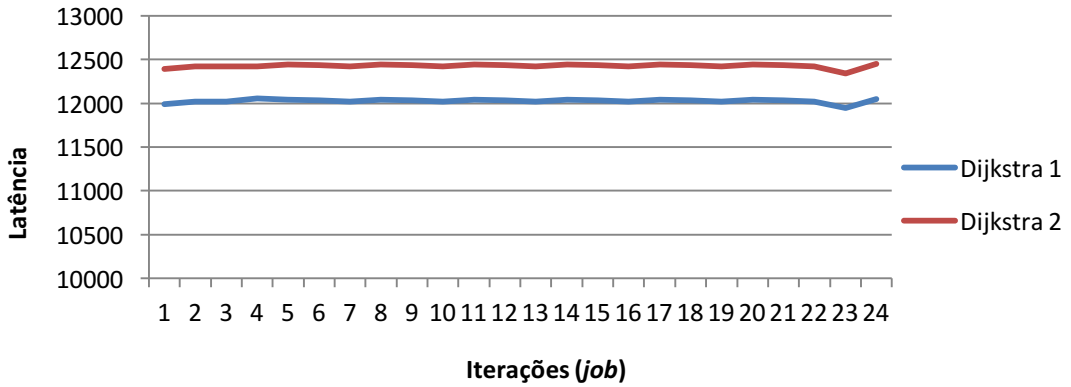


Figura 43 - Variação de latência de comunicação de duas aplicações RT em MPSoC 8x8 com 1 tarefa por PE e 80% de ocupação.

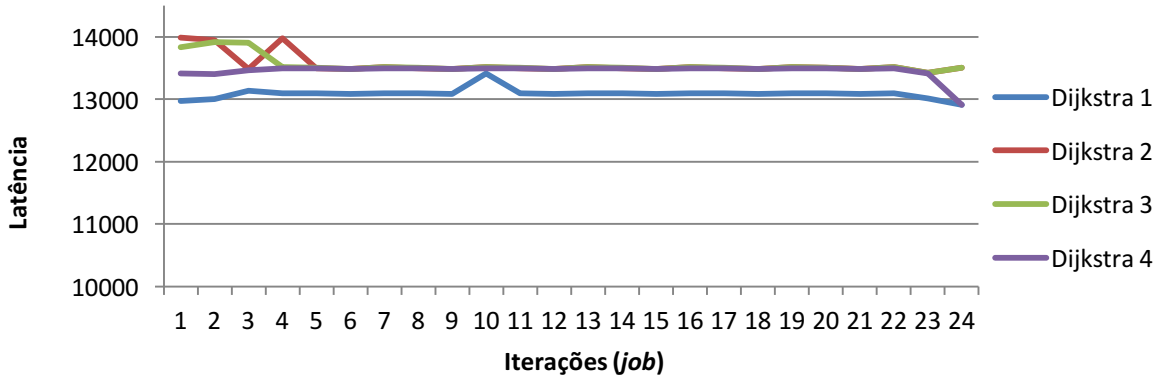


Figura 44 - Variação de latência de comunicação de quatro aplicações RT em MPSoC 8x8 com 3 tarefa por PE e 50% de ocupação.

Dessa forma, pode-se concluir que, independente da aplicação e configuração do MPSoC, a variação da latência de comunicação das aplicações com restrições de tempo-real possui uma latência com baixo *jitter* (desvio padrão), o que contribui para o atendimento às restrições das aplicações.

6.5 Estudo de Caso

Esta Seção apresenta uma análise dos mecanismos apresentados na Tese, com a utilização de um benchmark baseado nos Veículos Autônomos [SHI10B]. Este benchmark é modelado de forma sintética mas baseado em dados reais apresentado por Indrusiak [IND14]. O benchmark é composto de 51 tarefas comunicantes que desempenham funcionalidades como: controle de vibração automotivo, detecção de obstáculos e controle de navegação.

O benchmark de veículos autônomos trabalha como um conjunto de aplicações. A Figura 45 apresenta o grafo de tarefas do benchmark, mostrando o nome da tarefa e sua prioridade.

A Tabela 21 apresenta os principais dados do benchmark, definindo as restrições de computação de cada tarefa e o volume de comunicação de cada par comunicante. A computação de cada tarefa é o WCET dos *jobs* da tarefa. O período de uma determinada tarefa é o tempo que o conjunto de tarefas ao qual ela pertence precisa para finalizar uma iteração. A prioridade da tarefa define a sua execução quando compartilha processamento com outra tarefa. As tarefas finais de cada aplicação desempenham a saída dos dados para depuração, por isso não apresentam restrições de tempo-real. A computação da funcionalidade da aplicação é de responsabilidade das demais tarefas.

A modelagem do benchmark de Veículos Autônomos foi feita utilizando a linguagem de programação C. A comunicação entre as tarefas é programada conforme API de comunicação disponibilizada pela plataforma HeMPS (MPI embarcado).

Como apresentado no Capítulo 5, o teste de escalonabilidade define se a configuração do MPSoC possibilita a execução das aplicações no sistema. Para analisar se o MPSoC atende as restrições de tempo das tarefas, o benchmark de veículos autônomos foi testado nas configurações dos cenários apresentados na Tabela 22.

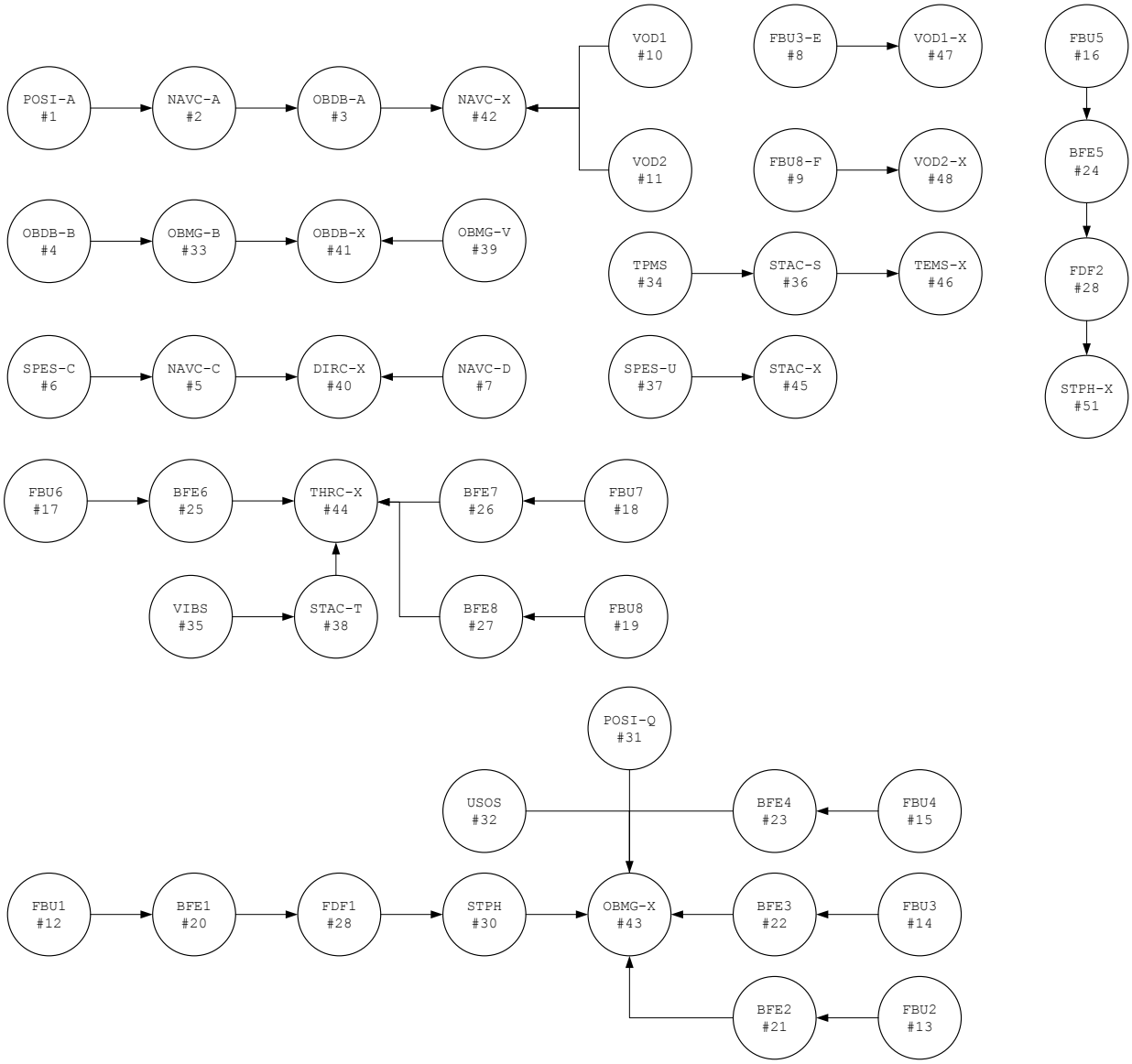


Figura 45 – Grafo de tarefas do benchmark Veículos Autônomos. Fonte: [IND14]

Tabela 21 – Benchmark de Veículos Autônomos.

Identificador	Nome	Tarefa Comunicante	Computação (ms)	Período (ms)	Prioridade	Comunicação (byte)
1	POSI-A	2	5	500	31	2048
2	NAVC-A	3	10	500	32	4096
3	OBDB-A	42	150	500	33	32768
4	OBDB-B	33	150	1000	34	65536
5	NAVC-C	40	20	100	24	1024
6	SPES-C	5	5	100	25	1024
7	NAVC-D	40	10	100	26	2048
8	FBU3-E	47	10	40	1	76800
9	FBU8-F	48	10	40	2	76800
10	VOD1	42	20	500	3	1024
11	VOD2	42	20	500	4	1024
12	FBU1	20	10	40	5	76800
13	FBU2	21	10	40	6	76800
14	FBU3	22	10	40	7	76800
15	FBU4	23	10	40	8	76800
16	FBU5	24	10	40	9	76800
17	FBU6	25	10	40	10	76800
18	FBU7	26	10	40	11	76800
19	FBU8	27	10	40	12	76800
20	BFE1	28	20	40	13	4096
21	BFE2	43	20	40	14	4096
22	BFE3	43	20	40	15	4096
23	BFE4	43	20	40	16	4096
24	BFE5	29	20	40	17	4096
25	BFE6	44	20	40	18	4096
26	BFE7	44	20	40	19	4096
27	BFE8	44	20	40	20	4096
28	FDF1	30	10	40	21	16384
29	FDF2	51	10	40	22	16384
30	STPH	43	30	40	23	8192
31	POSI-Q	43	5	500	35	2048
32	USOS	43	5	100	27	2048
33	OBMG-B	41	20	1000	37	8192
34	TPMS	36	5	500	36	4096
35	VIBS	38	5	100	28	1024
36	STAC-S	46	10	1000	38	4096
37	SPES-U	45	5	100	29	2048
38	STAC-T	44	10	100	30	2048
39	OBMG-V	41	0,5	1000	39	4096
40	DIRC-X	–	–	–	–	–
41	OBDB-X	–	–	–	–	–
42	NAVC-X	–	–	–	–	–
43	OBMG-X	–	–	–	–	–
44	THRC-X	–	–	–	–	–
45	STAC-X	–	–	–	–	–
46	TPMS-X	–	–	–	–	–
47	VOD1-X	–	–	–	–	–
48	VOD2-X	–	–	–	–	–
49	FDF1-X	–	–	–	–	–
50	FDF2-X	–	–	–	–	–
51	STPH-X	–	–	–	–	–

Fonte: [IND14]

Tabela 22 – Configuração dos cenários testes do benchmark de Veículos Autônomos.

Cenário	Tamanho do MPSoC	Tarefas por PE	Atende às Restrições de Tempo	Motivo
1	6x6	1	Não	O MPSoC contém apenas 35 recursos disponíveis, menos do que a aplicação requer (51).
2	6x6	>1	Não	O MPSoC contém a quantidade de recursos necessárias, mas a aplicação não pode compartilhar tantas tarefas quanto necessário.
3	7x7	1	Não	O MPSoC contém apenas 48 recursos disponíveis, menos do que a aplicação requer (51).
4	7x7	>1	Sim	O MPSoC atende aos requisitos de tempo da aplicação.
5	8x8	1	Sim	O MPSoC atende aos requisitos de tempo da aplicação.
6	8x8	>1	Sim	O MPSoC atende aos requisitos de tempo da aplicação.

A tabela acima mostra a adequação de projeto do MPSoC para que possa executar as tarefas atendendo suas restrições de tempo-real. Os cenários 1 e 3 não atendem às restrições de tempo das aplicações por não conter recursos suficientes para executar a aplicação. O cenário 2 contém os recursos necessários para mapear as aplicações, mas o teste de escalonabilidade, Equação 1, não encontra o mapeamento ideal. Os cenários 4, 5 e 6 atendem aos requisitos de tempo da aplicação. A Figura 46 apresenta o resultado do mapeamento da aplicação nos cenários apresentados.

Pode-se analisar que o mapeamento de tarefas baseado no modelo RTA permite o compartilhamento de CPU para os casos em que uma tarefa, considerando a interferência na computação de outra, não afetará seu desempenho. Tarefas com prioridades semelhantes ou com prazo de entrega curto tendem a não compartilhar a mesmo CPU.

Slave 0x6	Slave 1x6	Slave 2x6 spes_u 2304 TER	Slave 3x6 usos 1292 TER	Slave 4x6	Slave 5x6	Slave 6x6
Slave 0x5 tpms_x 2050 TER	Slave 1x5 stac_s 2048 TER	Slave 2x5 stph 1291 TER	Slave 3x5 fbu2 1285 TER	Slave 4x5 fbu8 1797 TER	Slave 5x5	Slave 6x5
Slave 0x4 stph_x 1539 TER	Slave 1x4 posi_q 1290 TER	Slave 2x4 fbu1 1284 TER	Slave 3x4 bfe2 1281 TER	Slave 4x4 fbu3 1286 TER	Slave 5x4 stac_t 1798 TER	Slave 6x4
Slave 0x3 fdf1 1288 TER	Slave 1x3 fbu3_e 768 TER vod1_x 769 TER	Slave 2x3 fbu8_f 1024 TER vod2_x 1025 TER	Slave 3x3 bfe1 1280 TER obmg_x 1289 TER	Slave 4x3 bfe3 1282 TER	Slave 5x3 fbu4 1287 TER	Slave 6x3 vibs 1800 TER
Slave 0x2 fbu5 1537 TER	Slave 1x2 vod1 4 TER	Slave 2x2 bfe4 1283 TER	Slave 3x2 obmg_v 259 TER	Slave 4x2 navc_d 514 TER	Slave 5x2 bfe7 1793 TER	Slave 6x2 fbu7 1796 TER
Slave 0x1 bfe5 1536 TER obdb_a 2 TER	Slave 1x1 navc_a 0 TER navc_x 1 TER	Slave 2x1 vod2 5 TER thrc_x 1799 TER	Slave 3x1 obdb_b 256 TER obdb_x 257 TER	Slave 4x1 dirc_x 512 TER	Slave 5x1 spes_c 515 TER	Slave 6x1 bfe8 1794 TER
Global M 0x0	Slave 1x0 tpms 2049 TER posi_a 3 TER	Slave 2x0 stac_x 2305 TER fdf2 1538 TER	Slave 3x0 obmg_b 258 TER	Slave 4x0 navc_c 513 TER	Slave 5x0 bfe6 1792 TER	Slave 6x0 fbu6 1795 TER

(a) Cenário 4

Slave 0x7 spes_u 2304 TER	Slave 1x7 thrc_x 1799 TER	Slave 2x7 stac_x 2305 TER	Slave 3x7	Slave 4x7	Slave 5x7	Slave 6x7	Slave 7x7
Slave 0x6 stac_t 1798 TER	Slave 1x6 fbu8 1797 TER	Slave 2x6 vibs 1800 TER	Slave 3x6 posi_q 1290 TER	Slave 4x6	Slave 5x6	Slave 6x6	Slave 7x6
Slave 0x5 fbu7 1796 TER	Slave 1x5 bfe8 1794 TER	Slave 2x5 obmg_x 1289 TER	Slave 3x5 fbu1 1284 TER	Slave 4x5 stph 1291 TER	Slave 5x5	Slave 6x5	Slave 7x5
Slave 0x4 bfe7 1793 TER	Slave 1x4 fdf1 1288 TER	Slave 2x4 bfe4 1283 TER	Slave 3x4 bfe2 1281 TER	Slave 4x4 fbu3 1286 TER	Slave 5x4 usos 1292 TER	Slave 6x4 fdf2 1538 TER	Slave 7x4
Slave 0x3 vod1_x 769 TER	Slave 1x3 fbu3_e 768 TER	Slave 2x3 fbu8_f 1024 TER	Slave 3x3 bfe1 1280 TER	Slave 4x3 bfe3 1282 TER	Slave 5x3 fbu4 1287 TER	Slave 6x3 bfe5 1536 TER	Slave 7x3 stph_x 1539 TER
Slave 0x2 vod2 5 TER	Slave 1x2 posi_a 3 TER	Slave 2x2 vod2_x 1025 TER	Slave 3x2 obmg_b 258 TER	Slave 4x2 fbu2 1285 TER	Slave 5x2 navc_d 514 TER	Slave 6x2 fbu5 1537 TER	Slave 7x2
Slave 0x1 navc_x 1 TER	Slave 1x1 navc_a 0 TER	Slave 2x1 vod1 4 TER	Slave 3x1 obdb_b 256 TER	Slave 4x1 obmg_v 259 TER	Slave 5x1 dirc_x 512 TER	Slave 6x1 spes_c 515 TER	Slave 7x1 tpms 2049 TER
Global M 0x0	Slave 1x0 obdb_a 2 TER	Slave 2x0 bfe6 1792 TER	Slave 3x0 obdb_x 257 TER	Slave 4x0 fbu6 1795 TER	Slave 5x0 navc_c 513 TER	Slave 6x0 stac_s 2048 TER	Slave 7x0 tpms_x 2050 TER

(b) Cenário 5

Slave 0x7 fbu8 1797 TER	Slave 1x7 bfe8 1794 TER	Slave 2x7 stac_t 1798 TER	Slave 3x7	Slave 4x7	Slave 5x7	Slave 6x7	Slave 7x7
Slave 0x6 bfe7 1793 TER	Slave 1x6 bfe6 1792 TER thrc_x 1799 TER	Slave 2x6 fbu6 1795 TER	Slave 3x6 vibs 1800 TER	Slave 4x6 stac_s 2048 TER tpms 2049 TER	Slave 5x6 spes_u 2304 TER stac_x 2305 TER	Slave 6x6	Slave 7x6
Slave 0x5 fbu7 1796 TER	Slave 1x5 usos 1292 TER	Slave 2x5 fbu3 1286 TER	Slave 3x5	Slave 4x5 tpms_x 2050 TER	Slave 5x5	Slave 6x5	Slave 7x5
Slave 0x4 stph 1291 TER	Slave 1x4 fbu2 1285 TER	Slave 2x4 bfe3 1282 TER	Slave 3x4 fbu4 1287 TER	Slave 4x4	Slave 5x4 fdf2 1538 TER	Slave 6x4	Slave 7x4
Slave 0x3 fbu1 1284 TER	Slave 1x3 fbu8_f 1024 TER vod2_x 1025 TER	Slave 2x3 bfe1 1280 TER obmg_x 1289 TER	Slave 3x3 bfe4 1283 TER	Slave 4x3 fdf1 1288 TER	Slave 5x3 bfe5 1536 TER	Slave 6x3 stph_x 1539 TER	Slave 7x3
Slave 0x2 posi_q 1290 TER	Slave 1x2 vod1 4 TER	Slave 2x2 bfe2 1281 TER	Slave 3x2 obmg_v 259 TER	Slave 4x2 navc_d 514 TER	Slave 5x2 fbu5 1537 TER	Slave 6x2	Slave 7x2
Slave 0x1 obdb_a 2 TER	Slave 1x1 navc_a 0 TER navc_x 1 TER	Slave 2x1 vod2 5 TER	Slave 3x1 obdb_b 256 TER obdb_x 257 TER	Slave 4x1 dirc_x 512 TER	Slave 5x1 spes_c 515 TER	Slave 6x1 fbu3_e 768 TER vod1_x 769 TER	Slave 7x1
Global M 0x0	Slave 1x0 posi_a 3 TER	Slave 2x0	Slave 3x0 obmg_b 258 TER	Slave 4x0 navc_c 513 TER	Slave 5x0	Slave 6x0	Slave 7x0

(c) Cenário 6

Figura 46 – Mapeamento de tarefas para aplicação de Veículos Autônomos.

7 CONCLUSÕES E TRABALHOS FUTUROS

Esta Tese apresentou uma adequação de modelos arquiteturais para aplicações com restrições de tempo-real em sistemas multiprocessados. A adequação dos modelos deu-se pela análise dos dados de diferentes arquiteturas de memória. A modelagem abstrata de plataformas deste tipo de sistema permite efetuar sua avaliação e verificar as limitações de *software* e *hardware* presentes na arquitetura. Heurísticas de mapeamento e escalonamento são chaves para o atendimento às restrições de tempo impostas pelas aplicações.

O Capítulo de Introdução da Tese apresentou as seguintes hipóteses: *"(i) modelagem de arquiteturas abstratas podem ser utilizadas para avaliar limitações de hardware para sistemas RT; e (ii) soluções de mapeamento de tarefas com base em modelos analíticos e algoritmos de escalonamento de tarefas podem definir o sistema como hard ou soft RT."*

A extensiva avaliação apresentada no trabalho apresentou as limitações de *hardware* do sistema, utilizando modelos de arquiteturas em alto nível de abstração. As avaliações possibilitaram a adequação e definição da arquitetura para a continuidade da pesquisa. Com base no modelo adequado de MPSoC, fez-se uma avaliação de heurísticas de mapeamento e escalonamento. Tais avaliações possibilitaram a análise das heurísticas propostas na Tese.

O Anexo A apresenta as publicações, do Autor desta Tese, relacionadas ao trabalho (7 artigos em conferências e 2 artigos em periódicos).

7.1 Conclusões Relacionadas a Modelagens Abstratas

A modelagem de arquiteturas abstratas demonstrou-se eficiente na avaliação das limitações de *hardware* para sistemas RT. Tais modelagens apresentaram as limitações de um sistema baseado em memória compartilhada. Concluiu-se que o uso de plataformas com organização de memória compartilhada gera tráfego localizado (*hotspot*) induzindo o congestionamento da rede. Além do congestionamento, podemos levar em consideração o tempo de vida dos componentes do sistema. A área do *hotspot* tende a ter seu tempo de vida reduzido comparado às demais áreas do sistema. Dessa maneira, a área afetada pode apresentar falhas em seus componentes diminuindo os recursos disponíveis para computação e podendo impossibilitar o acesso à memória compartilhada. Com a modelagem abstrata de sistemas baseados em arquitetura de memórias distribuídas notou-se a não existência de *hotspots*, com um tráfego uniformemente distribuído pelo MPSoC. O volume relativo de comunicação é muito semelhante entre os roteadores, mostrando uma melhor distribuição do tráfego de dados que em sistemas baseados em memória compartilhada.

Os modelos projetados em alto nível de abstração, apresentados na Tese, expõem as vantagens e limitações de cada sistema. Também, mostram os benefícios da utilização de modelagens abstratas para auxiliar os projetistas na tomada de decisões de projetos em sistemas

embarcados. Os modelos de alto nível de abstração, apesar de não serem precisos em ciclo de relógio, permitem uma estimativa rápida de dados com baixo erro (<17%) no início do projeto em parâmetros como tempo de execução, volume de dados trafegado na NoC, número de instruções executadas em cada processador. O número de instruções executadas permite estimar a energia consumida pelo sistema.

7.2 Conclusões Relacionadas às Heurísticas de Mapeamento

Esta Tese propôs uma heurística de mapeamento guiada por um método analítico de escalonabilidade. O método analítico RTA avalia a interferência de um conjunto de tarefas, executando no mesmo processador, em outra tarefa. Esta interferência define se uma tarefa pode ter seu tempo de execução alterado a ponto de não conseguir atender às restrições de tempo-real estipuladas pela aplicação.

Nos testes avaliados, notou-se que o mapeamento baseado no método analítico RTA garante o atendimento de 100% dos *deadlines* em todos os casos avaliados. Os resultados apresentaram que os dados resultantes das aplicações com restrições de tempo aconteceram dentro do período de tempo definido. Comparado com as heurísticas de mapeamento HEAT e LEC-DN, o RTA demostrou ser o mapeamento indicado para sistemas com restrições *hard real-time*.

7.3 Conclusões Relacionadas aos Mecanismos de Escalonamento

Na presente Tese, implementou-se o mecanismo de escalonamento preemptivo por prioridade. Este algoritmo prioriza a execução da tarefa com maior prioridade, só podendo ser interrompida caso exista alguma tarefa com maior prioridade. Para efeito comparativo, o mecanismo de escalonamento LST foi implementado e avaliado como mecanismo de referência nos casos de teste.

Considerando que sistemas com restrições de tempo são caracterizados em *soft-RT* e *hard-RT*, concluiu-se que o mecanismo de escalonamento PP é o indicado para sistemas *hard real-time*. Esta conclusão é devido ao fato deste escalonador atender a todas as restrições de tempo impostas pelas aplicações. Por outro lado, o escalonador LST pode ser utilizado em sistemas com restrições *soft real-time*, onde em alguns momentos as violações são aceitáveis.

7.4 Limitações da Proposta

As principais limitações da Tese são:

- A heurística de mapeamento é executada em tempo de projeto;
- Os testes de mapeamento foram feitos com base em mapeamento com diferentes abordagens, visando fazer um comparativo com alto e baixo nível de compartilhamento de processadores.

7.5 Trabalhos Futuros

As limitações citadas na Seção anterior não invalidam os resultados obtidos e analisados, mas podem indicar possíveis direções a serem seguidas em pesquisas futuras. Dessa maneira, os trabalhos futuros incluem:

- Avaliar outros mecanismos de escalonamento (por exemplo, round-robin baseado em prioridades, rate-monotonic, EDF);
- Estender a heurística de mapeamento RTA para mapear tarefas em tempo de execução;
- Desenvolver novos benchmarks com dados de tempo-real;
- Avaliar outras funções custo para mapeamento, encontradas no estado da arte como: análise térmica e MCM;
- Avaliar o mecanismo de escalonamento RTA para os critérios do HEAT e do LEC-DN (consumo energético de processamento e comunicação).

REFERÊNCIAS

[AGU08] Aguiar, A; Johann, S.; Santos, T.; Marcon, C.; Hessel, F. *Architectural Support for Task Migration Concerning MPSoC*. In: Workshop de Sistemas Operacionais (WSO), 2008, pp. 169-178.

[ALM09] Almeida, G.; Sassatelli, G.; Benoit, P.; Saint-Jean, N.; Varyani, S.; Torres, L.; Robert, M. *An Adaptive Message Passing MPSoC Framework*. International Journal of Reconfigurable Computing, vol. 2009, 2009, 20p.

[AMI12] Amin, M.; Tagel, M.; Jervan, G.; Hollstein, T. *Design Methodology for Fault-Tolerant Heterogeneous MPSoC under Real-Time Constraints*. In: International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2012, 6p.

[AUD93] Audsley, N.; Burns, A.; Richardson, M.; Tindell, K.; Wellings, A. *Applying New Scheduling Theory to Static Priority Preemptive Scheduling*. Software Engineering Journal, vol. 8(5), 1993, pp. 284-292.

[AUS02] Austin T.; Larson E.; Ernst D. *SimpleScalar: An Infrastructure for Computer System Modeling*. Computer, vol. 35(2), 2002, pp. 59-67.

[BAR11] Barthe, L.; Cargnini, L.; Benoit, P.; Torres, L. *The SecretBlaze: A Configurable and Cost-Effective Open-Source Soft-Core Processor*. In: International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011, pp. 310-313.

[BEN12] Benini, L.; Flamand, E.; Fuin, D.; Melpignano, D. *P2012: Building an Ecosystem for a Scalable, Modular and High-Efficiency Embedded Computing Accelerator*. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012, pp. 983-987.

[BER04] Bertozzi, D.; Benini, L. *Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip*. Circuits and Systems Magazine, vol. 4(2), 2004, pp. 18-31.

[BES12] Beserra, G.S.; Attarzadeh Niaki, S.H.; Sander, I. *Integrating Virtual Platforms into a Heterogeneous MoC-based Modeling Framework*. In: Forum on Specification and Design Languages (FDL), 2012, pp. 143-150.

[BIN11] Binkert, N.; Beckmann, B.; Black, G.; Reinhardt, S.; Saidi, A.; Basu, A.; Hestness, J.; Hower, D.; Krishna, T.; Sardashti, S.; Sen, R.; Sewell, K.; Shoaib, M.; Vaish, N.; Hill, M.; Wood, D. *The GEM5 Simulator*. Computer Architecture News, vol. 39(2), 2011, 7p.

[BJE04] Bjerregaard, T.; Sparso, J. *Virtual Channel Designs for Guaranteeing Bandwidth in Asynchronous Network-on-Chip*. In: Norchip Conference, 2004, pp. 269-272.

[BOL04] Bolotin, E.; Cidon, I.; Ginosar, R.; Kolodny, A. *QNoC: QoS Architecture and Design Process for Network-on-Chip*. Systems Architecture, vol. 50(2), 2004, pp. 105-128.

[BON15] Bonilha, I.; Santos, O.; Indrusiak, L. *Heuristics for Mapping Real-Time Applications to NoC-Based Architectures Using Genetic Algorithms*. In: Brazilian Symposium on Computing Systems Engineering (SBESC), 2014, pp. 144-149.

[BOU07] Boukhechem, S.; Bourennane, E.; Smahi, A. *Co-simulation Platform Based on SystemC for Multiprocessor System on Chip Architecture Exploration*. International Conference on Microeletronics (ICM), 2007, pp. 105-110.

[BUS11] Busseuil, R.; Barthe, L.; Almeida, G.; Ost, L.; Bruguier, F.; Sassatelli, G.; Benoit, P.; Robert, M.; Torres, L. *Open-Scale: A Scalable, Open-Source NoC-based MPSoC for Design Space Exploration*. In: Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011, pp. 357-362.

[CAS13] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F. *Distributed Resource Management in NoC-based MPSoCs with Dynamic Cluster Sizes*. In: Computer Society Annual Symposium on VLSI (ISVLSI), 2013, pp. 153-158.

[CAS16] Castilhos, G.; Mandelli, M.; Ost, L.; Moraes, F. *Hierarchical Energy Monitoring for Task Mapping in Many-core Systems*. Journal Science Architecture, vol. 63, 2016, pp. 80-92.

[CHA08] Chantem, T.; Dick, R.; Sharon, X. *Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs*. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2008, pp. 288-293.

[CHA99] Chang, H.; Cooke, L.; Hunt, M.; Martin, G.; McNelly, A.; Todd, L. *Surviving the SoC Revolution: A Guide to Platform-based Design*. Kluwer Academic Publishers, 1999, 235p.

[CLA12] Clauss, C.; Pickartz, S.; Lankes, S.; Bemmerl, T. *Towards a Multicore Communications API Implementation (MCAPI) the Intel Single-Chip Cloud Computer (SCC)*. In: International Symposium on Parallel and Distributed Computing (ISPDC), 2012, pp. 148-155.

[COR15] CORBA Object Management Group. *CORBA Specifications*. Capturado em: http://www.omg.org/spec/index.htm, 2015.

[DAG98] Dagum, L.; Menon, R. *OpenMP: An Industry Standard API for Shared-Memory Programming*. Computational Science & Engineering, vol. 5(1), 1988, pp. 46-55.

[DAN06] Dangui, S. *Modelagem e Simulação de Barramentos com SystemC*. Dissertação de Mestrado, Instituto de Computação, Unicamp, 2006, 114p.

[DCO15] DCOM Microsoft Corporation. *Understanding the DCOM Wire Protocol by Analyzing Network Data Packets*. Capturado em: www.microsoft.com/msj/0398/dcom.aspx, 2015.

[DOR05] Dorta, A.; Rodruiguez, C.; Sande, F.; Gozales-Escribano, A. *The OpenMP Source Code Repository*. In: Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP), 2005, pp. 244-250.

[DUE14] Duenha, L.; Guedes, M.; Almeida, H.; Boy, M.; Azevedo, R. *MPSoCBench: A Toolset for MPSoC System Level Evaluation*. In: International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2014, pp. 164-171.

[FID08] Fide, S.; Jenks, S. *Architecture Optimizations for Synchronization and Communication on Chip Multiprocessors*. In: International Symposium on Parallel and Distributed Processing (IPDPS), 2008, 8p.

[GHE06] Ghenassia, F. *Transaction Level Modeling with SystemC*. Springer US, 2005, 271p.

[GOO03] Goossens, J.; Funk, S.; Baruah, S. *Priority-driven Scheduling of Periodic Task Systems on Multiprocessors*. Real-time Systems, vol. 25(2), 2003, pp. 187-205.

[GUT01] Guthaus, M.R.; Ringenberg, J.S.; Ernst, D.; Austin, T.M.; Mudge, T.; Brown, R.B. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In: Proceedings of the Workload Characterization (WWC), 2001, pp. 3-14.

[HEN88] Hensgen, D.; Finkel, R.; Manber, U. *Two Algorithms for Barrier Synchronization". International Journal of Parallel Programming*, vol. 17(1), 1988, 17p.

[HER10] Herveille, R. *WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*. Technical Report OpenCores Organization, 2010, 128p.

[HUN10] Hung, S.; Yang, W.; Chia-Hen, T. *Designing and Implementing a Portable, Efficient Inter-core Communication Scheme for Embedded Multicore Platforms*. In: International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010, pp. 303-308.

[IBS13] International Business Strategies, Inc (IBS), 2013.

[IND14] Indrusiak, L. *End-to-end Schedulability Tests for Multiprocessor Embedded Systems based on Networks-on-Chip with Priority-Preemptive Arbitration*. Journal of Systems Architecture, vol. 60(7), 2014, pp. 553-561.

[INT15] Intel Inc. *Intel® MPI Benchmarks 4.0 Update 2*. Capturado em: software.intel.com/en-us/articles/intel-mpi-benchmarks, 2015.

[IQB10] Iqbal, S.; Liang, Y.; Grahn, H. *ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems*. In: Computer Architecture Letters (CAL), 2010, pp. 45-48.

[JOV08] Joven, J.; Font, O.; Castells, D.; Martinez, R.; Teres, L.; Carrabina, J. *xENoC – An eXperimental Network-on-Chip Environment for Parallel Distributed Computing on NoC-based MPSoC Architecture*. In: Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP), 2008, pp. 141-148.

[JOV13] Joven, J.; Marongiu, A.; Angiolini, F.; Benini, F.; De Micheli, G. *An Integrated, Programming Model-Driven Framework for NoC-QoS Support in Cluster-based Embedded Many-Core*. Parallel Computing, v.39(10), 2013, pp. 549-566.

[KAH77] Kahn, G.; MacQueen, D. *Coroutines and Networks of Parallel Programming*. In: IFIP Conference on Optimization Techniques, 1977, pp. 993-998.

[KLE07] Klein, F.; Araujo, G.; Azevedo, R.; Leao, R.; Santos, L. *An Efficient Framework for High-Level Power Exploration*. In: Midwest Symposium on Circuits and Systems (MWSCAS), 207, pp. 1046-1049.

[KOP97] Kopetz, H. *Real-Time Systems: Design Principle for Distributed Embedded Applications.* Kluwer Academic, Norwell, 1997.

[KUM13] Kumar, S.; Djie, M.; Leuken R. *Low Overhead Message Passing for High Performance Many-Core Processors*. In: International Symposium on Computing and Networking (CANDAR), 2013, pp. 345-351.

[LEM12] Lemaire, R.; Thuries, S.; Heiztmann, F. *A Flexible Modeling Environment for a NoC-Based Multicore Architecture*. In: International High Level Design Validation and Test Workshop (HLDVT), 2012, pp. 140-147.

[LI03] Li, J; Yao, C. *Real-Time Concepts for Embedded Systems*. CPM Books, 2003, 294p.

[LIU13] Liu, D.; Spasic, J.; Chen, G. *Energy-Efficient Mapping of Real-Time Streaming Applications on Cluster Heterogeneous MPSoCs*. In: Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), 2013, 10p.

[LOG04] Loghi, M.; Angiolini, F.; Bertozzi, D.; Benini, L.; Zafalon, R. *Analyzing on-Chip Communication in a MPSoC Environment*. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2004, pp. 752-757.

[LUK08] Lukovic, S.; Fiorin, L. *An Automated Design Flow for NoC-Based MPSoCs on FPGA*. In: International Symposium on Rapid System Prototyping (RSP), 2008, pp. 58-64.

[MAD15] Madalozzo, G.; Mandelli, M.; Ost, L.; Moraes, F. *A Platform-based Design Framework to Boost Many-Core Software Development.* In: International Conference on Electronics and Communication Systems (ICECS), 2015, pp. 320-323.

[MAD16A] Madalozzo, G.; Duenha, L.; Azevedo, R.; Moraes, F. *Scalability Evaluation in Many-core Systems due to the Memory Organization*. In: International Conference on Electronics and Communication Systems (ICECS), 2016, pp. 396-399.

[MAD16B] Madalozzo, G.; Indusiak, L.; Moraes, F. *Mapping of Real-Time Applications in a Packet Switchinf NoC-based MPSoC.* In: International Conference on Electronics and Communication Systems (ICECS), 2016, pp. 640-643.

[MAH08] Mahr, P.; Lorchner, C.; Ishebabi, H.; Bobda, C. *SoC-MPI: A Flexible Message Passing Library for Multiprocessor Systems-on-Chips*. In: Conference on Reconfigurable Computing and FPGAs (ReConFig), 2008, pp. 187-192.

[MAN13] Mandelli, M.; Rosa, F.; Ost, L.; Sassatelli, G.; Moraes, F. *Multi-level MPSoC Modeling for Reducing Software Development Cycle*. In: International Conference on Electronics and Communication Systems (ICECS), 2013, pp. 489-492.

[MAN15] Mandelli, M.; Ost, L. ; Sassatelli, G., Moraes, F. *Trading-off system load and communication in mapping heuristics for improving NoC-based MPSoCs reliability*. In: International Symposium on Quality Electronic Design (ISQED), 2015, pp. 392-396.

[MAR06] Sepúlveda, M. *Estimativa de Desempenho de uma NoC a partir de seu Modelo em SystemC-TLM*. Dissertação de Mestrado, USP, 2006, 172p.

[MAR11] Marongiu, A.; Burgio, P.; Benini, L. *Supporting OpenMP on a Multi-Cluster Embedded MPSoC*. Microprocessors and Microsystems, vol. 35(8), 2011, pp. 668-682.

[MAT10A] Mattson, T.; Wijngaart, R.; Riepen, M.; Lehnig, T.; Brett, P.; Haas, W.; Kennedy, P.; Howard, J.; Vangal, S.; Borkar, N.; Ruhl, G.; Dighe, S. *The 48-core SCC Processor: The Programmer's View*. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2010, 11p.

[MAT10B] Mattson, T.; Wijngaart, R.; *RCCE: A Small Library for Many-Core Communication*. Technical Report Intel Corporation, 2010.

[MAT11] Matilainen, L.; Salminen, E.; Hamalainen, T.D.; Hannikainen, M. *Multicore Communications API (MCAPI) Implementation on an FPGA Multiprocessor*. In: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2011, pp. 286-293.

[MCM96] McMahon, T.; Skjellum, A. *eMPI/eMPICH: Embedding MPI*. In: MPI Developer's Conference (MPIDC), 1996, pp. 180-184.

[MEI10] Meier, M.; Engel, M.; Steinkamp, M.; Spinczyk, O. *LavA: An Open Platform for Rapid Prototyping of MPSoC*. In: International Conference on Field Programmable Logic and Applications (FPL), 2010, pp. 452-457.

[MIC13] MicroBlaze Processor. *MicroBlaze Processor Reference Guide: EDK14*. Relatório Técnico, 2013, 254p.

[MIN09] Minhass, W.; Öberg, J.; Sander, I. *Design and Implementation of a Plesiochronous Multi-core 4x4 Network-on-chip FPGA with MPI HAL Support*. In: FPGAworld Conference (FPGAworld), 2009, pp. 52-57.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. *HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*. The VLSI Journal, vol. 38(1), 2004, pp. 69-93.

[MPI15] MPI. *The Message Passing Interface (MPI) Standard*. Capturado em: www.mcs.anl.gov/research/projects/mpi/, 2015.

[OPE15] OpenMP API. *The OpenMP API Specification for Parallel Programming*. Capturado em: www.openmp.org/, 2015.

[ORT09] Ortiz, A.; Indruziak, L.; Murgan, T.; Glesner, M. *PMD: A Low-Power Code for Networks-on-Chip based on Virtual Channels*. Integrated Circuit and Systems Design: Power and Timing Modeling, Optimization and Simulation, vol. 5349, 2009, pp. 219-228.

[OST11] Ost, L.; Guindani, G.; Moraes, F.; Indrusiak, L.; Määttas, S. *Exploring NoC-based MPSoC Design Space with Power Estimation Models*. Design & Test of Computer, vol. 28(2), 2011, pp. 16-29.

[OST12] Ost, L.; Varyani, S.; Indrusiak, L.; Mandelli, M.; Almeida, G.; Wachter, E.; Moraes, F.; Gilles, S. *Enabling Adaptive Techniques in Heterogeneous MPSoCs Based on Virtualization*. Transactions on Reconfigurable Technology and Systems, vol. 5(3), 2012, 11p.

[OVP15] OVP Imperas. *Open Virtual Platform (OVP)*. Capturado em: www.ovpworld.org/, 2015.

[PAU04] Paulin, P. *Application of a Multi-Processor SoC Platform to High-Speed Packet Forwarding*. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2004, pp. 58-63.

[PAU06] Paulin, P.; Pilkington, C.; Langevin, M.; Bensoudane, E.; Lyonnard, D.; Benny, O.; Lavigueur, B.; Lo, D.; Beltrame, G.; Gagné, V.; Nicolescu, G. *Parallel Pragramming Models for a Multiprocessor SoC Platform Applied to Networking and Multimedia*. Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14(7), 2006, pp. 667-680.

[PET12] Petry, C.; Wachter, W.; Castilhos, G; Moraes, G.; Calazans, N. *A Spectrum of MPSoC Models for Design Space Exploration and its Use*. In: International Symposium on Rapid System Prototyping (RSP), 2012, pp. 30-35.

[RAM09] Ramming, F.; Ditze, M.; Janacik, P. *Basic Concepts of Real-Time Operating Systems*. In: HDS, 2009, pp. 15-45.

[REI09] Reinbrecht, C.; Scartezzini, G.; Raupp, T. *Desenvolvimento de um Ambiente de Execução de Aplicações Embarcadas para a Plataforma Multiprocessada HeMPS*. TCC, FACIN, PUCRS, 2009.

[REK13] Rekik, W.; Ben, M.; Ben, N. *Virtual Prototyping of Multiprocessor Architectures Using the Open Virtual Platform*. In: International Conference on Computer Applications Technology (ICCAT), 2013, 6p.

[RIG03] Rigo, S.; Araujo, G.; Bartholomeu, M.; Azevedo, R. *ArchC: A SystemC-based Architecture Description Language*. In: Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2004, pp. 66-73.

[ROS05] Rose, A.; Swan, S.; Pierce, j.; Fernandez, J. *Transaction Level Modeling in SystemC*. Open SystemC Initiative, 2005.

[ROS15] Rosa, R.; Lemaire, R.; Clermidy, F. *A Co-Design Approach for Hardware Optimizations in Multicore Architectures using MCAPI*. In: International Workshop on Interconnection Network Architectures: On-Chip, Multi-Chi (INA-OCMC), 2015, 4p.

[RUA14] Ruaro, M.; Carara, E.; Moraes, F. *Tool-set for NoC-based MPSoC Debugging – a Protocol View Perspective*. In: International Symposium on Circuits and Systems (ISCAS), 2014, pp. 2531-2534.

[RUA15] Ruaro, M.; Madalozzo G.; Moraes F. *A hierarchical LST-based task scheduler for NoC-based MPSoCs with slack-time monitoring support*. In: International Conference on Electronics and Communication Systems (ICECS), 2015, pp. 308-311.

[RUA16] Ruaro, M.; Lazzarotto, F.; Marcon, C.; Moraes, F. *DMNI: A Specialized Network Interface for NoC-based MPSoCs*. In: International Symposium on Circuits and Systems (ISCAS), 2016, pp. 1202-1205.

[SAL06] Saldana, M.; Chow, P. *TMD-MPI: An MPI Implementation for Multiple Processors across Multiple FPGAs*. In: International Conference on Field Programmable Logic and Applications (FPL), 2006, pp.1-6.

[SAN04] Sander, I.; Jantsch, A. *System modeling and transformational design refinement in ForSyDe*. Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23(1), 2004, pp. 17-32.

[SAN13] Santos, F. *MediaBox: Uma Plataforma Baseado em NoCs para Aplicações Multimídia*. Dissertação de Mestrado, Instituto de Computação, Unicamp, 2013, 77p.

[SHI10A] Shi, Z.; Burns, A. *Schedulability Analysis and Task Mapping for Real-Time on-Chip Communication*. Real-Time Systems, vol. 46(3), 2010, pp. 360-385.

[SHI10B] Shi, Z.; Burns, A.; Indrusiak, L. *Schedulability Analysis for Real Time on-Chip Communication with Wormhole Switching*. Journal of Embedded and Real-Time Communication Systems, vol. 1(2), 2010, 22p.

[SIL12] Silva, V.; Fontes, C.; Wagner, F. *The Impact for Synchronization in Message Passing while Scaling Multi-Core MPSoC Systems*. In: International Conference on VLSI and System-on-Chip (VLSI-SoC), 2012, pp. 267-270.

[SIM15] Simics. *Simics Specification*. Capturado em: www.windriver.com/products/simics, 2015.

[SIN13] Singh, A.; Shafique, M.; Kumar, A.; Henkel, J. *Mapping on Multi/Many-core Systems: Survey of Current and Emerging Trends*. In: Design Automation Conference (DAC), 2013, 10p.

[SOM10] Sommerville, I. “Software Engineering”. Addison-Wesley Publishin Company, 2010, 9° edição, USA.

[STO10] Stone, J.; Gohara, D.; Guochun, S. *OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems*. Computing in Science & Engineering, vol. 12(3), 2010, pp. 66-73.

[THO10] Thonnart, Y.; Vivet, P.; Clermidy, F. *A Fully-Asynchronous Low-Power Framework for GALS NoC Integration*. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, pp. 33-38.

[TIA09] Tian, G.; Hammami, O. *Performance Measurements of Synchronization Mechanisms on 16PE NoC-based Multi-Core with Dedicated Synchronization and Data NoC*. In: International Conference on Electronics and Communication Systems (ICECS), 2009, pp. 988-991.

[VAH01] Vahid, F.; Givardis, T. *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons Inc. vol. 1, 2001, 352p.

[WAN08] Wang, Z.; Hammami, O. *A Twenty-four Processors System on Chip FPGA Design with on-Chip Network Connection*. In: IP SoC, 2008, 4p.

[WON07] Wong, H.; Rendell, A. *The Design of MPI Based Distributed Shared memory Systems to Support OpenMP on Clusters*. In: International Conference on Cluster Computing (CLUSTR), 2007, pp. 231-240.

[WOO95] Woo, S.C.; Ohara, M.; Torrie, E.; Singh, J.P.; Gupta, A. *The SPLASH-2 Programs: Characterization and Methodological Considerations*. In: International Symposium on Computer Architecture (AISCA), 1995, pp. 24-36.

[WOS07] Woszezenki, C. *Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs*. Dissertação de Mestrado, FACIN, PUCRS, 2007.

[XIL15] Xilinx. *ML403 Evaluation Platform Documentation*. Capturado em: www.xilinx.com/products/boards/ml403/docs.htm, 2015.

[YOO03] Yoo, S.; Jerraya, A. *Introduction to Hardware Abstraction Layer for SoCs*. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013, pp. 336-337.

[ZAY13] Zaykov, P.; Kuzmanov, G.; Molnos, A.; Goossens K. *Run-Time Slack Distribution for Real-Time Data-Flow Applications on Embedded MPSoC*. In: Euromicro Conference on Digital System Design (DSD), 2013, pp. 39-47.

[ZHA09] Zhaoguo, S.; Chaoshan, L.; Zuying, L. *An Task Scheduling Algorithm of Real-Time Leakage Power and Temperature Optimization for MPSoC*. In: International Conference on Computer-Aided Design and Computer Graphics (CADCG), 2009, pp. 478-483.

[ZHA13] Zhang, D.; Zeng, X.; Wang, Z.; Wang, W.; Chen, X. *MCVP-NoC: Many-Core Virtual Platform with Networks-on-Chip Support*. In: International Conference on ASIC (ASICON), 2013, pp. 28-31.

[ZHO16] Zhou, J.; Wei, T.; Chen, M.; Yan, J.; Hu, X.; Ma, Y. *Thermal-Aware Task Scheduling for Energy Minimization in Heterogeneous Real-Time MPSoC Systems*. Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35(8), 2016, pp. 1269-1282.

[ZHU10] Zhuo, L.; Du, G.; Zhang, D.; Song, Y.; Li, L.; Pan, H. *Design and Implementation of DDR2 Wrapper for Cluster based MPSoC*. In: International Conference on Anti-Counterfeiting, Security and Identification (ICASID), 2010, pp. 60-62.

ANEXO A – PUBLICAÇÕES DO AUTOR

A Tabela 23 apresenta o conjunto de publicações do Autor no período do Doutorado. As publicações 4, 6 (artigos em periódico), 7 (artigo em periódico) e 8 foram efetuadas em cooperação com o Laboratório de Sistemas Computacionais (LSC) do Instituto de Computação da Universidade de Campinas (IC-Unicamp). A publicação 9 foi feita em cooperação com o Instituto de Informática da Universidade de York.

Tabela 23 – Publicações do Autor no período do Doutorado.

	Publicações	Descrição
1	**Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes** CASTILHOS, G.; MANDELLI, M.; MADALOZZO, G.; MORAES, F. In: ISVLSI, 2013	Capítulo 3
2	**A Framework for MPSoC Generation and Distributed Applications Evaluation** CASTILHOS, G.; WACHTER, E.; MADALOZZO, G.; ERICHSEN, A.; MONTEIRO, T.; MORAES, F. In: ISQED, 2014	
3	**A Hierarchical LST-Based Task Scheduler for NoC-Based MPSoCs with Slack-Time Monitoring Support** RUARO, M.; MADALOZZO, G.; MORAES, F. In: ICECS, 2015	Capítulo 6
4	**Exploração de Desempenho, Consumo Dinâmico e Eficiência Energética em MPSoC** DUENHA, L.; MADALOZZO, G.; SANTIAGO, T.; MORAES, F.; AZEVEDO, R In: WSCAD, 2015	Capítulos 3 e 4
5	**A Platform-Based Design Framework to Boost Many-Core Software Development** MADALOZZO, G.; MANDELLI, M.; OST, L.; MORAES, F. In: ICECS, 2015	
6	**MPSoCBench: A Benchmark for High-level Evaluation of Multiprocessor System-on-Chip Tools and Methodologies** DUENHA, L.; MADALOZZO, G.; SANTIAGO, T.; MORAES, F.; AZEVEDO, R Journal of Parallel and Distributed Computing, 2016	
7	**Exploiting Performance, Dynamic Power and Energy Scaling in Full-system Simulators** DUENHA, L.; MADALOZZO, G.; MORAES, F.; AZEVEDO, R. Concurrency and Computation: Practice and Experience, 2016	
8	**Scalability Evaluation in Many-Core Systems due to the Memory Organization** MADALOZZO, G.; DUENHA, L.; AZEVEDO, R.; MORAES, F. In: ICECS, 2016	
9	**Mapping of Real-Time Applications on a Packet Switching NoC-based MPSoC** MADALOZZO, G.; INDRUSIAK, L.; MORAES, F. In: ICECS, 2016	Capítulo 5 e 6